(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2001－306191
（P2001－306191A）
(43)公開日　平成13年11月２日(2001.11.2)

(51)Int.Cl.'		識別記号	ＦＩ		テーマコード'(参考)
Ｇ０６Ｆ	1/26		Ｈ０２Ｊ	1/00	３０４Ｄ　５Ｂ０１１
Ｈ０２Ｊ	1/00	３０４	Ｇ０６Ｆ	1/00	３３４Ｈ　５Ｇ０６５

審査請求　未請求　請求項の数10　ＯＬ　（全 29 頁）

(21)出願番号　特願2000－119033(P2000－119033)

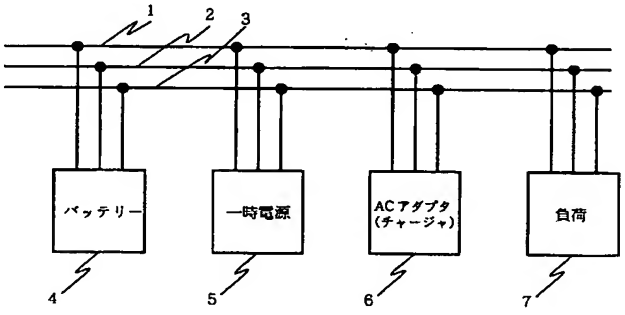
(22)出願日　平成12年４月20日(2000.4.20)

(71)出願人　000002185
ソニー株式会社
東京都品川区北品川６丁目７番35号
(72)発明者　田島　茂
東京都品川区東五反田３丁目14番13号　株式会社ソニーコンピュータサイエンス研究所内
(74)代理人　100101801
弁理士　山田　英治　（外２名）

Ｆターム(参考)　5B011 DA02 DA11 DA12 DA13 DB21 DB26 DC06 GG02 HH02 JB10
5G065 EA02 EA04 PA05

(54)【発明の名称】　電源サーバ、電源クライアントおよび電源バスシステム

(57)【要約】
【課題】　電源や負荷を統合的な環境で利用できるようにする。
【解決手段】　バッテリ４、一時電源５、ＡＣアダプタ６、負荷７が電源バスライン１、接地バスライン２およびデータバスライン（バスシステム）に接続されている。バスシステムに接続されている各ブロックは自らをオブジェクトとして記述し、各ブロックのオブジェクトが、データバスラインを介して相互に状態データ等のやり取りを行う。各ブロックのオブジェクトは、他のブロックのオブジェクトからの要求に基づいて状態データを生成し、これを回答する。回答データを受け取ったブロックのオブジェクトはこれに基づいて電源の供給や消費を制御する。



1

【特許請求の範囲】
【請求項１】　電源バス接続ポートと、
データバス接続ポートと、
当該データバス接続ポートより電源クライアントの状態を取得する手段と、
当該電源バス接続ポートを介して電力を供給する手段とを有することを特徴とする電源サーバ。
【請求項２】　電源バス接続ポートと、
データバス接続ポートと、
当該データバス接続ポートに対して状態を送信する手段と、
当該電源バス接続ポートを介して電力を受け取る手段とを有することを特徴とする電源クライアント。
【請求項３】　電源サーバと、
電源クライアントと、
電源サーバから電源クライアントに対して電源を供給する電源バスとを具備し、
当該電源クライアントは、
電源バス接続ポートと、
データバス接続ポートと、
当該データバス接続ポートに対して状態を送信する手段と、
当該電源バス接続ポートを介して電力を受け取る手段とを有し、
当該電源サーバは、
電源バス接続ポートと、
データバス接続ポートと、
当該データバス接続ポートより電源クライアントの状態を取得する手段と、
当該電源バス接続ポートを介して電力を供給する手段とを有することを特徴とする電源バスシステム。
【請求項４】　電源バス接続ポートと、
データバス接続ポートと、
当該データバス接続ポートを伝送されるデータを受け取る手段と、
受け取った上記データに基づいて電力の消費を制御する手段とを有することを特徴とする電源消費ブロック。
【請求項５】　電源バス接続ポートと、
データバス接続ポートと、
当該データバス接続ポートを伝送されるデータを受け取る手段と、
受け取った上記データに基づいて電力の供給を制御する手段とを有することを特徴とする電源供給ブロック。
【請求項６】　少なくとも１つの電源供給ブロックと、
少なくとも１つの電源消費ブロックと、
電源バスおよびデータバスを具備し上記電源供給ブロックから上記電源消費に対して電源を供給するバスシステムとを有し、
上記電源消費ブロックは、
電源バス接続ポートと、

2

データバス接続ポートと、
上記データバス接続ポートを介してデータを送受信する手段と、
上記データバス接続ポートを介して受け取ったデータに基づいて上記電源バスを介して送られる電力の消費を制御する手段とを有し、
上記電源供給ブロックは、
電源バス接続ポートと、
データバス接続ポートと、
上記データバス接続ポートを介してデータを送受信する手段と、
上記データバス接続ポートを介して受け取ったデータに基づいて上記電源バスへの電源の供給を制御する手段とを有することを特徴とする電源システム。
【請求項７】　上記電源消費ブロックおよび電源供給ブロックの属性がデータおよび関数として定義され、これらデータが上記データバスを介して上記電源供給ブロックおよび電源消費ブロックの間で送受信される請求項６記載の電源システム。
【請求項８】　上記電源消費ブロックおよび電源供給ブロックの少なくとも一部が上記バスシステムに対して着脱可能である請求項６または７記載の電源システム。
【請求項９】　上記電源消費ブロックおよび電源供給ブロックの属性が、データとその操作アルゴリズム（関数）、継承、ポリモーフィズムを用いて記述される請求項６、７または８記載の電源システム。
【請求項１０】　上記バスシステムはデータバス管理の為にアドレス信号を周期的に発生し、そのアドレス信号に対応する情報は、上記バスシステムに接続されるブロックが発信する請求項６、７、８または９記載の電源システム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】この発明は、種々の電源や負荷を統合的に管理する技術に関する。
【０００２】
【従来の技術】バッテリー等を電源とする、ポータブル機器の発展にともない、複数のバッテリー搭載機器を同時に使用する機会が増加しつつある。例えば、モバイルコンピュータの画像をＨＭＤ（ヘッドマウンテッド・ディスプレイ）で観測する場合や、モバイルコンピュータをモデムカードを仲介として携帯電話に接続する場合などである。この様な場合においても、現在のところモバイルＰＣ（パーソナル・コンピュータ）にはそれ専用の、またＨＭＤや携帯電話にはそれら専用のバッテリーが用いられ、お互いの間でのエネルギーの融通は無い。
【０００３】また、ＰＣやその周辺機器の発展に伴いウエアラブルＰＣが提唱され、いくつかの商品化例も見られる様になった。この場合でも、電源に関しては専用に設計されたものが使用され、例え手近に満充電された二

次電池があろうとも互換性が無いため流用できない。

【０００４】一方、ウエアラブルＰＣを含むモバイル機器では、いかに電源を確保するかというのは切実な問題であり、場合によっては手回し発電機の様な機器さえ必要になる。

【０００５】しかしながら、現在では、各種電源や各種負荷（機器）の間で統一的に電源を制御するアーキテクチャは無く、二次電池やＡＣアダプタ、簡易発電機等を設計するにしてもある特定の機器専用となってしまう。

【０００６】

【発明が解決しようとする課題】本発明は、上記に述べた不具合を解決するため、電源供給源と電源消費者という考えに基づいて、それらを抽象化し、統一的に定義し、複数の電源供給源（バッテリーやＡＣアダプタ等）、複数の電源消費者が一つの共通バスライン上に柔軟に着脱可能となるシステムを提供することを目的としている。

【０００７】

【課題を解決するための手段】本発明によれば、上述の目的を達成するために、特許請求の範囲に記載のとおりの構成を採用している。ここでは、特許請求の範囲の記載について補足的に説明をしておく。

【０００８】すなわち、本発明によれば、電源消費ブロックや電源供給ブロックに、電源バス接続ポートと、データバス接続ポートと、当該データバス接続ポートを伝送されるデータを受け取る手段と、受け取った上記データに基づいて電力の消費や電源の供給を制御する手段とを設けている。

【０００９】電源消費ブロックの中には、電源供給をあわせて行えるものがあってもよい。電源供給ブロックの中に電源消費をあわせて行えるものがあってもよい。

【００１０】この構成においては、各ブロックは送受信するデータに基づいて最適な状態で（予め設定された基準の基づく）電源の供給や消費を行え、また、バスシステムを用いて統合的に電源供給・消費やデータの送受信を行える。したがって、種々のブロック間で電源供給・消費について融通し会うことができる。

【００１１】また、バスシステムが、データバス管理のためにアドレス信号を周期的に発生し、そのアドレス信号に対応する情報は、バスに接続されるブロックが発信するようにしてもよい。

【００１２】この場合、発生するアドレス信号が、システム上可能な最大数までとすることが好ましい。

【００１３】また、発生するアドレス信号が、バス上にあるブロックの数＋１としてもよい。

【００１４】単一のデータバスラインを用いて、それに接続される複数のブロックのアドレスの設定および、データ通信の双方を行うようにすることが好ましい。

【００１５】また、単一のデータバスラインを用いてそれに接続される複数のブロックのアドレスの設定および、データ通信を二つの段階に分けて行うようにしてもよい。

【００１６】単一のデータバスラインに複数のブロックが接続され、それら複数のブロックのアドレス付与の方式が以下の特徴を持つことが好ましい。アドレス設定開始タイミングがすべてのブロックに対して同時（半導体素子のばらつきや回路のディレイ等による差はここでは含まない）であり、各ブロックはランダムに発生したデータを同じタイミングでデータバスに出力し、従って、データバスライン上にはそれらのオア信号が発生し、データバスライン上の情報と各ブロックが発生した信号を比較する事で、他のブロックが発生したデータを知る事なく、一つのブロックが特定でき、特定されたブロックにユニークなアドレスを割り当て、この一連の動作を繰り返す事で最終的にはすべてのブロックにユニークなアドレスが割り当てられる。

【００１７】また、アドレス付与後に、以下に示すユニークアドレス確認を行うことが好ましい。データバス管理手段が発生するアドレスに対し、そのアドレスの固有データをデータバス上に出力する事で、オア信号を発生し、各ブロックが自分の固有データとバス上のデータを照合する事で、自分以外に同じアドレスが存在する事を認識し、同じアドレスが存在するときには、再度、同位置のアドレスを有するブロックの範囲で、上述のアドレスバス付与を行ないアドレスを振り直す。

【００１８】また、データバスライン上の特定アドレスに対するブロック属性が消滅した事でブロックの削除を認識し、逆にブロック属性が存在していなかった特定アドレスに対して、ブロック属性を新規検出する事で、ブロックの接続を認識するようにしてもよい。さらに、バスラインに接続された任意のブロックの間で、双方向のデータ通信が可能であり、かつ、この通信により任意のブロックが（自身を含む）任意のブロックの制御、情報読み出し、情報かき込みが可能なようにすることが好ましい。

【００１９】

【発明の実施の形態】以下、本発明について詳細に説明する。

【００２０】本発明は、ハードウエアとして、電源およびデータバスとそのバスコントローラ、それらバスに接続される電源構成ブロック、ソフトウエアとして電源構成ブロックのオブジェクトモデル化および、バスコントローラの動作メカニズム、電源構成ブロック間の通信フォーマットを包含する一つのアーキテクチャに関するものである。

【００２１】ここでは内容が多岐にわたるため、次の３部構成で説明する。第１部：電源構成ブロックをどの様にオブジェクトモデル化するかについて説明。第２部：電源オブジェクトモデルを使用して実際の電源バスシステムをいかに構築するかについて説明する。第３部：構

5

築された電源バスシステム上で各オブジェクトモデルがどの様に動作するかについて説明する。

【００２２】［第１部　電源に使用されるブロックのオブジェクトモデル化］

１．電源に使用されるブロックの整理

現在電源関係に使用されている、あるいは可能性のあるブロックは概略以下のものである。

（１）　電源供給源

：一次電池（乾電池等）

：ＡＣアダプタ（商用ラインからＤＣを発生させ、機器の電源または二次電池の充電に用いる）

：ＤＣアダプタ（カーバッテリ等のＤＣ電源からターゲット機器に適するＤＣを発生させ、ＡＣアダプタと同様の用途に用いる）

：簡易発電機（エンジン付の発電機から、手動発電機まで）

：太陽電池

：空間の電磁波からエネルギーを発生するもの（例えば、特開平１０－１４６０７７号公報）

：その他各種燃料電池、生体からエネルギーを取り出すもの

（２）　電源消費者（負荷）

：各種機器（一般に、この部分が実ユーザに対して何らかの利便性を提供する）

（３）　電源供給、消費源

：二次電池（ＮｉＣｄ電池、Ｎｉ水素電池、リチウム電池、リチウムイオン電池、鉛電池等）

（電動自転車におけるモータの様に通常は電力を消費するものだが、制動時に電力発生の可能性をもつブロック（回生制動機能をもつもの）は、後述する理由により（３）ではなく（２）負荷に分類するのが適当である。）

つまり、電源関係で使用されるブロックを、高いレベルで抽象化すると上記３つにわけられ、さらに（３）は（１）と（２）の機能を共通にもつものと考えられる。またこれらを抽象化するには主としてコンピュータプログラミング分野で用いられている考えであるオブジェクトが有効である。従って、以下上記（１）、（２）、（３）をオブジェクトとして定義していくが、最初に簡単にオブジェクトとは何かについて説明する。

【００２３】２．オブジェクトの説明

オブジェクトとは、通常はコンピュータのプログラミング手法で用いられる用語であって、ある機能を実現する部分をブラックボックスとして扱い、ユーザ（プログラマ）にはその外部から見た仕様のみを与え、プログラミングの効率化を図る事を主目的とする。

【００２４】オブジェクトは、その機能ブロックの仕様を示すデータ、そしてそれらのデータを変更したり、何らかの機能を実現する関数群から構成される。

【００２５】オブジェクト指向そのものは特定のプログラム言語専有のものではないが、ここではもっとも一般的なＣ＋＋言語により説明する。オブジェクトはＣ＋＋においてはクラスとして定義され、例えば次の様なものである。（以下、山下　他著、Ｃ＋＋プログラミングスタイル、ｐ１４７、オーム社より引用）

【表１】

```
class Turtle{
public :
   int x;
   int y;
   int direction ;
   void forward(int);
   void turnleft();
   void turnright();
};

void Turtle::forward(int step)
{
  //前進の処理
}
```

以下略

（引用終了）

この例では、Ｔｕｒｔｌｅ（亀）クラスを定義し、そのデータとして位置座標ｘ、ｙと方向ｄｉｒｅｃｔｉｏｎ、クラスに対する動作を定義するものとして関数ｆｏｒｗａｒｄ（ｉｎｔ），ｔｕｒｎｌｅｆｔ（），ｔｕｒｎｒｉｇｈｔ（）を定義している。つまり、Ｔｕｒｔｌｅというオブジェクトを、その現在位置、進行方向、左回り動作、右回り動作、前進動作というデータおよび機能で規定している。

【００２６】３　電源に使用されるブロックのモデル化

（３－１）電源システムに使用されるブロックの抽象化

図１は一般的な電源の供給、消費システムを示したもので、１、２、３はそれぞれ電源、ＧＮＤ、信号のためのバスラインである。３の信号バスについては、ＧＮＤ２を共通帰線とする単一の線の場合、電源ラインとは独立に複数の線により構成される場合、さらに電源バス（１）に重畳され物理的には存在しない場合もある。４はバッテリー（ニッケル水素やリチウムイオンの様な二次電池、マンガンやアルカリ等の一次電池を問わない）、５は電気二重層コンデンサを貯蔵源とする様な一時電源、６はＡＣアダプタ（チャージャと兼用の場合もある）、７は負荷となる一般電気機器である。これらバスに接続されるもの（以降でオブジェクトとして定義する）の数は問わない。いずれのブロックもパワーバスラインに必ずしもすべてが同時に物理的に接続されるものではなく、各ブロックとパワーバスラインとの間にスイッチ（実用的には半導体スイッチ）が挿入されている。

【００２７】電源システムを構成する要素としては上記の様に各種考えられるが、これらの要素を抽象化すると、電源供給源となるもの（一次、ＡＣアダプタ、ＤＣアダプタ等）および、これらから電源を供給される消費者（負荷）に大別できる。オブジェクト指向の特徴の一つは、共通機能を持つ新しいオブジェクトをその元となるオブジェクトから派生させることが可能な点にある。従って、ここでもこのフィロソフィーに従い、まず本電源バスシステムに接続されるオブジェクトを定義し、それより電源供給源オブジェクトと負荷としてのオブジェクトを派生する。ＡＣアダプタや一次電池などは電源供給源オブジェクトから派生させる。電気機器そのものは、本システムからみれば単に負荷オブジェクトとして抽象化される。また１で述べた（３）項に相当するものは電源供給源、および負荷の機能を継承して生成される。実用機器で主要な電源となる二次電池はこの（３）項に属すると考えるのが適当である。

【００２８】また、二次電池に対する充電方式はそのバッテリーの種類（ＮｉＣｄかリチウムイオンか等）により異なるし、電気二重層コンデンサを用いた一時電源の充電の仕方はバッテリーとも異なる。しかし、これらの相違点もオブジェクト指向の大きなフィロソフィーであるポリモーフィズムにより統一的に扱え、単にユーザ（この場合は、本アーキテクチャに基づいて電源関係機器を設計する設計者）からは同一のインターフェースで扱える。

【００２９】（３－２）電源オブジェクト、負荷オブジェクトの投入、切断

図１に示すシステムにおいて、電源供給オブジェクト（図２）、負荷オブジェクト（図４）とも電源バスラインに対してスイッチを有している。電源供給オブジェクト、負荷オブジェクトとも１個の時は、それぞれに電源に対するスイッチを有するのは冗長であるが、複数の電源オブジェクトや負荷オブジェクト、さらには、電源または負荷となるオブジェクトを考慮するとこういう構造

にするのが適当である。

【００３０】この様な構造で、いったい誰がどのオブジェクトの（電源バスに対する）スイッチを投入するかについて以下の様に規定する。電源供給オブジェクトのスイッチは、複数の電源供給オブジェクトが同じバスライン上に存在するときに、それらのうち一つのみを有効にするために用いる。複数の電源供給オブジェクトから一つを選択する際には、次のプライオリティー付けに従う。

【００３１】

１　ＡＣアダプタ

２　ＤＣアダプタ

３　二次電池

４　一次電池

５　一時エネルギーストレージ

即ち、最も長時間エネルギーが供給できそうなものの順とする。また、同じレベルの電源供給オブジェクトが存在する場合は、例えばその容量順とかアドレス順とする（これはシステムの実装上の問題である）。負荷オブジェクトのスイッチ投入は、負荷の先に存在するセットの要求により制御されるのが原則であるので、負荷自身が制御する。この時、制御命令を自分自身に対して発信するようにする。こうすれば、バスライン上のほかのオブジェクトも情報共有ができる。

【００３２】４　電源供給源ブロックのオブジェクトモデル化

（４－１）最上位オブジェクトモデル

以下、本実施例に示す電源システムに接続されるブロックのオブジェクトモデル化について説明する。

【００３３】最初に、本電源システムに接続されるすべてのもの（バス管理手段を除く）を含むクラスを定義し、すべてのオブジェクトはこれから派生ないし継承で生成する。従って、このモデルがすべてのクラスの原型となる。

【表２】

```
typedef enum { POWER_SOURCE, LOAD, SOURCE_OR_LOAD } BlockType;
  //電源ブロックを、供給源、負荷、供給源または負荷 の３種類に定義する。
typedef enum { POWER_SOURCE, LOAD } PowerMode ;
  //電源のモード定義。 供給源 または 負荷。
typedef enum { FALSE, TRUE } boolean ;
  //boolean の定義。
class Load;
class LoadList{                  //  電源供給されている負荷リスト
public:
   Load* load ;
   LoadList* next ;
}

class PowerBlock {
private :
   BlockType block_type ;        //  電源供給ブロック、負荷、電源供給または負荷 を示すデータ
   PowerMode power_mode ;        //  電源供給モードか負荷モードかを示すデータ
   int address ;                 //  ブロックのユニークなアドレス
   boolean switch ;              //  オブジェクトの電源バススイッチ状態、F=オフ、T=オン
public :
   BlockType BlockTypeInq(){ return block_type ;}   //  BlockType 問い合わせ関数
   PowerMode ModeInq(){ return power_mode ;}        //  PowerMode 問い合わせ関数
   boolean SwitchCont( boolean cont){ return switch ;} // スイッチ制御関数
   boolean SwitchInq(){ return switch ;}        // スイッチ状態問い合わせ関数
};
```

上記クラスの定義の中で、データとしてアドレスは存在するが、関数としてアドレスを読み出すものはない。これは、本システムの構造上、オブジェクトに対してアドレスを問い合わせる事はできない（つまり、データ問い合わせのベースとなるのがアドレスであるので）からである。

【００３４】（４－２）電源供給源のモデル化

バッテリーやＡＣアダプタ等の電源供給源の基本モデルとなるものである。このモデルは電源供給源として必要な情報、およびそれに関連する機能をまとめたものである。

【００３５】図２において、１０はＤＣ電源であり、これがバッテリーであるかＡＣを整流した直流電源であるかは問わない。１１は電流検出のための抵抗、１２は電力バス（１６）との接続を行うためのスイッチ、１９はこの物理装置を論理的なオブジェクトとするためのマイクロプロセッサ（即ち、情報バスライン１７から見込むとこの物理装置は論理的オブジェクトに見える）であり、具体的にはＤＣ電源の電圧や出力電流の測定、スイッチ１２のコントロール、自分自身のバス上のアドレス設定や他のオブジェクトとデータ通信を行う。

【００３６】この様な一般的な電源供給源のオブジェクトモデル例を以下に示す。

【表３】

11　　12

```
typedef int PowerType ;
class PowerSource : public PowerBlock {
   private:
      // 仕様
      PowerType power_type ;   // 電源の種類（一次電池、二次電池、AC アダプタ、DC アダプタ等）
      int catalog_voltage ;    // 公称出力電圧
      int max_current ;        // 公称最大出力電流
      int max_va ;             // 公称容量
      int alarm_voltage ;      // 電源最低供給可能電圧
      // 現在の状態
      int output_voltage ;     // 現在の出力電圧
      int output_current ;     // 現在の出力電流
      int current_imp ;        // 現在の内部インピーダンス
      int service_time ;       // 負荷の要求に対応できる予測時間
      int min_service_time ;   // 負荷の要求に対応する最低時間
      // 内部で使用するデータ
      LoadList *load_list ;    // 負荷リスト
   public:
      // 内部データを読み出すための関数
      PowerType PowerType(){ return power_type ; }
      int CatalogVoltage(){ return catalog_voltage ; }
      int MaxVA(){ return max_va ; }
      int AlarmVoltage(){ return alarm_voltage ; }
      int OutputVoltage(){ return output_voltage ; }
      int OutputCurrent(){ return output_current ;}
      int CurrentImp(){ return current_imp ; }
      int ServiceTime(){ return service_time ; }
   public :
      // オブジェクト制御のための関数
      boolean IsUsable( Load *l)   // 電源供給源が負荷オブジェクトにサービス可能か
      {
       引数は Load クラスへのポインターで、実質的に、負荷オブジェクトの、最低電圧、許容最高電圧、
       最小電流、最大電流等である。
       そして、関数としては、現在の電源供給源が
          負荷オブジェクトの許容最高電圧＞予想出力電圧 ＞負荷オブジェクトの要求最低電圧
          予想最大出力電流 ＞ 負荷オブジェクトの要求電流
          最低供給電圧（二次電池ならバッテリーダウン電圧）に達するまでの時間が、ある設定値以上か
       を評価し、False または True を返す。
      }
      boolean SetMinServTime( int t) // 最低サービスタイム設定
      {
       引数として設定時間を分単位でとり、正常設定なら True、エラーなら False を返す。 電源供給オブ
       ジェクトに対して、最低限の動作時間を設定。 これは主として一時ストレージ電源用コマンドであ
       る。
      }
      int RemainVA()        // 電源の残容量問い合わせ
      {
       電源の残容量でバッテリーな場合は電圧と電流より残容量を計算し、分単位で、AC アダプターや DC
       アダプターの場合ある決められた値（例えば０ｘｆｆ）を返す。
      }
};
```

また、以下に内部データに対するコードの割り当て例を示す。上記モデルだけでは二次電池は対象外であるが、二次電池は上記オブジェクトモデルと負荷オブジェクトモデルを継承して生成されるので、ＰｏｗｅｒＴｙｐｅのコード割り当てに二次電池や一時電源もいれておく。０ｘは１６進数を表す。

【表４】

PowerType		code
二次電池、	NiCd	0x00
	NiH	0x01
	LiIon	0x02
	Lead	0x03
一次電池	Mn	0x04
	Alkaline	0x05
AC アダプタ		0x06
DC アダプタ		0x07
発電機		0x08
一時電源		0x09

13　　14

【００３７】（４－３）　エネルギーチャージオブジェクト

これは、エネルギーストレージオブジェクトの充電、および電源供給を担当するものである。このオブジェクトはＡＣアダプターであってもいいし、カーバッテリからチャージする様なＤＣ入力のもの、あるいは、環境に存在する電磁波を電圧に変換する様な他の手段によるものでもいい。チャージャはそのターゲットとするものによってチャージの方式までの異なるので、それらを一まとめにした一般的なモデル例を以下に示す。実際のオブジェクトはこれからさらに派生させて詳細データを定義していく。またそれら個々のチャージャはその実現方式によって、使用するコマンドやパラメータが異なる（例えばＮｉＣｄ電池の充電においても、充電電圧の変化を検出する方式や、電池の温度変化を検出する方式、電池の状態を見ずに微少な電流で充電するいわゆるトリクル充電など種々の方式がある）ので、一般的にコマンド体系を定義するのは困難であるし、また適当ではない。そこで、この様な装置に固有なコマンドに対応するために、本発明報告のコマンド体系には、機器固有コマンドを送受するメカニズムが用意されている（第３部　７－３参照）。

【表５】

```
class Charger :public PowerSource{
    private :
        int charger_type ;          //  チャージャの型（Li イオン用、NiCd 用等の区別）
        int max_output_voltage ;    //  充電最大出力電圧
        int max_output_curr ;       //  最大充電電流
        int voltlim ;               //  出力電圧リミッタ値
        int currlim ;               //  出力電流リミッタ値
        boolean charge ;            //  充電中、非充電中
    public :
        int SetVoltLim( int v ){ return voltlim ; }     //  出力電圧リミッタ設定
        int SetCurrLim( int c ){ return currlim ; }     //  出力電流リミッタ設定
        boolean ChargeCont( boolean cont ){ return charge ;}  //  充電開始、終了
} ;
```

チャージャの型のコード割付例を示す。（ＰｏｗｅｒＴｙｐｅの割付例参照）

【表６】

charger_type	code
NiCd	0x00
NiH	0x01
LiIon	0x02
Lead	0x03
一時電源	0x09

【００３８】５．負荷オブジェクトのモデル化

（５－１）基本負荷オブジェクトのモデル化

基本的な負荷モデルに必要な情報および機能をまとめてモデル化する。実際のユーザにサービスを提供する製品は、本オブジェクト指向電源アーキテクチャにとっては負荷オブジェクトモデルの先に存在するもので、負荷オブジェクトの中に埋没し、それがどの様なものかは関与しない。

【００３９】図４において１６、１７、１８はそれぞれ電源バス、データバスおよびＧＮＤである。５０は本来の負荷５７（これが実際はユーザに利便性を提供する部分である）を入り切りするためのスイッチ、５５は負荷オブジェクトを生成するマイクロプロセッサである。消費電流は抵抗５１により検出する事を基本とするが、５７の電流消費状況は既知であるので、その５７を制御するマイクロプロセッサ５８より通信路５６を介してデータを入手してもいい。また、５５と５８が同一のプロセッサの場合も当然存在する。５９は電源バス１６の瞬断や、電源供給オブジェクトの交換時の急激な電圧変化を緩和するためのコンデンサで、電気二重層コンデンサや補助的二次電池等が使える。また省略可能な場合もある。

【００４０】以下にオブジェクト例を示す。

【表７】

15　　16

```
typedef int LoadType ;
typedef enum { LOAD_OFF, LOAD_SUSPEND, LOAD_ON } LoadMode ;
Class Load : public PowerBlock {
   private:
      LoadType load_type ;     // 負荷のタイプ、サスペンド可能等
      int max_voltage ;        // 最大動作可能電圧
      int min_voltage ;        // 最小動作可能電圧
      int max_curent ;         // 負荷 ON 時最大消費電流
      int min_current ;        // 負荷 ON 時最小消費電流
      int suspend_curr ;       // サスペンド時消費電流
      LoadMode load_mode ;     // 現在、負荷として接続されているか、否か。 サスペンドか
      int input_voltage ;      // 現在の入力電圧
      int input_current ;      // 現在の消費電流
   public:
      LoadType LoadTypeInq(){ return load_type ;}   // 負荷タイプ問い合わせ
      int MaxVoltage(){ return max_voltage ;}       // 最大負荷電圧要求
      int MinVoltage(){ return min_vltagae ;}       // 最小負荷電圧要求
      int MaxCurrent(){ return maz_current ;}       // 最大負荷電流要求
      int MinCurrent(){ return min_current ;}       // 最小負荷電流要求
      int SuspendCurrent(){ return suspend_current ;}  // サスペンド電流要求
      LoadMode LoadModeInq(){ return load_mode ;}      // 負荷モード要求
      int InputVoltage(){ return input_voltage ;}      // 現在の入力電圧問い合わせ
      int InputCurrent(){ return input_current ;}      // 現在の入力電流問い合わせ
      int GetVA() {                 // 負荷容量問いわせ
      引数 ； なし
      負荷の容量（電圧、電流）問い合わせ
      戻り値 ；  電圧  単位Ｖ  （上位４ビット）
                 電流 単位 １００ｍA  （下位４ビット）
      }
      int LoadCont( int cont){    // 負荷接続、切り離し
      引数 ；   0x00       負荷接続
                0x01       負荷切断
                0x02       負荷サスペンド
      戻り値 ； 0x00       正常動作
                0x01       エラー
      }
      int GetCurrPowerConsmpt(){  //現在の消費電力計算
      引数 ； なし
      戻り値 ； 電流Ｘ電圧 、単位 VA
      }
};
```

【００４１】（５－２）発電負荷オブジェクト
例えば、最終負荷としてモータを考える。モータがトルクを発生している特は、負荷オブジェクトは純粋に負荷であって、そこから電力が生まれてくる事はない。しかしながら、モータを外部より強制的に回転させる力が働くとき（例えば、電動自転車で、モータには電力供給をせずに、慣性力により逆にモータを駆動するとき）には、負荷オブジェクトが回生制動をサポートしているならば、このオブジェクトは電源供給オブジェクトとも成り得る。しかしながら、この様なオブジェクトの実用化には次の問題がある。

【００４２】即ち、電源バスシステムは、システムのアドレスポーリングタイミングに同期してのみ、電源供給オブジェクトの切り替えを行う事が可能で、システムに非同期に電力発生モードになる訳にはいかない。従って、該当オブジェクト内にこれを解決する手段が要るが、これは具体的には補助充電池や電気二重層コンデンサの様な蓄電手段を設けることになろう。しかもこの補助電源による電力はこのオブジェクト内で使用する様な構築とするのが実用的であるので、単に負荷オブジェクトとして定義するのが最適である。

【００４３】ただし、電源供給オブジェクト、負荷オブジェクトからの継承で生成できないと言う訳ではない。

【００４４】６．オブジェクトの継承
（６－１）二次電池のモデル化
二次電池や一時ストレージ等、ある時は電源供給源となるが、ある時は電力消費者（負荷）となるものは電源供給オブジェクトと負荷オブジェクトよりそれぞれの性質を継承して生成するのが適当である。

【００４５】図２において、ＤＣ電源１０をバッテリーと考えると、この図はそのままバッテリーオブジェクトの内部構成をとなる。この他にバッテリーの保護回路としてのヒューズや温度検出素子等、すでに実用化されている各種構成要素が含まれるが、本発明の趣旨に無関係

なので省略する。

【００４６】以下にオブジェクトモデル例を示すが、これ以外の定義の仕方も存在するし、また新データや新機

```
class RechargeBatt : public PowerSource, public Load {
    private:
       int ModelName ;          // バッテリーの型名
       int TotalLife ;          // バッテリーの寿命。 充放電回数。
       int TotalUsedTime ;      // 現在までの充電回数

    public :
       int GetLifeTime(){       // 寿命計算
       引数 ； なし
       バッテリの寿命を、残りの充放電回数として計算し、TotalUsedTime として保存
       戻り値 ； (TotalLife-TotalUsedTime)残り充電回数、 単位 ×１０回
       }
};
```

ここで、関数の中身（実装方式）は設計項目でありその具現化方式そのものは本発明とは直接の関係はなく、例えばバッテリー残量計算や表示アルゴリズムはすでにカムコーダなどで実現されている。

【００４７】バッテリーをこの様なクラスとして定義すると、負荷オブジェクトは以下の情報を知る事ができる。

｛１｝　現負荷に対してバッテリーが使用可能か

｛２｝　現在のバッテリーの残容量

｛３｝　現在使用しているバッテリーの（期待）寿命。

【００４８】また、バッテリーの型名等は負荷オブジェクトには興味ない事がらであるが、バスラインに電源ブロックエミュレータを接続すれば、各ブロックの情報を読み出せ、サービス等に役立てる事も可能となる。従って、これらの補助情報をオブジェクトに持たせる場合もある。

【００４９】（６－２）一時エネルギーストレージオブジェクト

これは、電気二重層コンデンサ等の大容量コンデンサを使用した、一時的に電圧を貯える事を主機能とするオブジェクトである。図３にブロック図を示すが、ＤＣ電源の部分が電気二重層コンデンサ２０に置き換わっただけで、他は図２と変わらない。マイクロプロセッサ１９は１４より電源が常に供給されるが、もちろん２０のに貯えられている電力が放電すれば動作を停止する。そのため２１の様に電力ラインから電力供給を受けるルートを用意しておく。また、電気二重層コンデンサ２０は１６を経由して充電されることも可能であるが、１６を経由せずに、全く別の経路で外部より充電される場合（例えば、電磁誘導により電力バス１６を介さずに）もある。

能をつけ加える事もある。

【表８】

【００５０】電気二重層コンデンサの容量は現在、０．１Ｆ程度から１００Ｆ程度が容易に入手できるが、この部分は用途によっては通常の電解コンデンサも使用可能であり、この時は容量は１０μＦ程度から、数千μＦであろう。従って、容量範囲が広いのでμＦを単位として、その係数を別途定義しておく。

【００５１】これをオブジェクトとして抽象化したものを次の例に示す。

【表９】

```
class TempStorage : public PowerSource, public Load {
   private :
     int capacitance ;     //    電気二重層コンデンサの容量、単位μF
     int multiplier ;      //    指数
   public :
     int Capacitance(){ return capacitance ;}
     int Multiplier(){ return multiplier ;}
};
```

【００５２】［第２部　電源オブジェクトモデルによる電源バスシステムの構築］

１．基本システムの構築

前項までで、電源の各ブロックのオブジェクトモデル化は完了したが、これを実際のバスライン上にインプリメントするためには、さらに次の様な機能を定義する必要がある。

【００５３】まず始めに、今回のバスシステム（電源、ＧＮＤ、データの３線式を仮定する）に電源の各ブロックを任意のタイミングで接続、切断を行う方式を説明する。

【００５４】図５において、６３はバスライン上に用意されたバス管理手段である。このバス管理手段は本報告でいうオブジェクトとは別のものである。この部分は実

際にはマイクロプロセッサで構成され、電源は電源バスから供給される。すなわち、バスライン上に電源供給オブジェクトが接続されない限り動作は開始しない。６１はデータラインのプルアップ抵抗、６２はマイクロプロセッサよりのアドレス管理データをデータバスに出力するためのもので、データラインは各電源オブジェクトとの間でワイアードオアとなっている。６４は電源オブジェクトの一例である（ここでは、データラインがワイアードオアとなっているのを図示するのが目的なので詳細は省略する）。

【００５５】バス管理手段６３は、それに電源が供給され動作を開始すると図６に示す様に、ある決まった数Ｎ（Ｎの詳細については第３部、１、アドレスフォーマット参照）のアドレスパルスを、サイクリックに発生する。アドレスパルスと次のアドレスパルスの間にはそれらのアドレスに対応するデータをデータバス上に出力する時間ｔが割り当てられている。図７は実際のデータパルス波形を示す。図ではアドレスに４ビット分を割り当てているが、最初は常に０で、アドレスのスタートを表現するため、最大アドレス数は８である。なお、アドレス数の最大はもちろん８に制限されないし、またバス管理手段にはリチウム電池等のバッテリーをあらかじめ装着しておいてもいい。

【００５６】以上の様な準備をしておいて、次に実際のアドレスの割り当ておよび通信の方式を示す。

【００５７】（１）アドレス設定モード

バスの状態には大きく分けて、各オブジェクトのアドレスを設定中の状態と、アドレスが確定してオブジェクト間でデータ通信を行っている状態がある。前者をアドレス設定モード、後者をデータ通信モードと定義する。アドレス設定モードは後述する様にアドレス設定プロセスとアドレス確認プロセスから構成される。

【００５８】アドレス設定モードは負荷オブジェクトがすでに接続された電源バスシステムに電源供給源オブジェクトが始めて接続された場合や、オブジェクトからエラー信号が発生され、再度電源システムの再構築が必要な場合に発生する。バスシステムがデータ通信モード中に新規オブジェクトが追加あるいは取り外しをされた場合には、アドレス設定はデータ通信モードの中で可能なので、アドレス設定モードとならない。（ただし、アドレス設定モードにする様にインプリメントする事はもちろん可能である。）詳細は後述する。

【００５９】（１－１）　アドレス設定プロセス

バスライン上に接続されたオブジェクトにアドレスが定義されていない状態で、電源が入りアクティブになったとき、最初に各オブジェクトにアドレスが定義される。これをアドレス設定プロセスと呼ぶ事にする。

【００６０】ここで実施しようとしているのは、全く同じタイミングに全く同じアルゴリズムを持った各オブジェクトが、一本のデータバスラインを共有しながら、それぞれを区別しようとするものである。このためには、共通のバスラインに何個のオブジェクトが接続されているかの判定と、それらの区別が必要である。これを例えばアナログ的に実施するならば、プルアップされたデータラインに各オブジェクトが同一のインピーダンスでプルダウンし、データラインのＤＣレベルを観測する方式が考えられる。（すなわち、オブジェクトの数に対応したＤＣレベルが観測される。）これを認識し、オブジェクトを一つずつ切り離していき、最後に残ったものに、まず一つのアドレスを与え、その既存アドレスを得たオブジェクト以外のオブジェクトに対して同じ事を繰り返す。この場合でも、同一アルゴリズムを持ったオブジェクトでは、どれを切り離すかの判定が（各オブジェクト自身）できない。そのため、各オブジェクトに乱数を発生させる機能を用意し、その値によってどのオブジェクトを切り離すか（あるいは残すか）をオブジェクト自身に判断させる。つまり、オブジェクト間で一種じゃんけんをして決める方式である。

【００６１】このような考えを、完全にディジタル的に実施したものを以下に述べる。

【００６２】ここでは、最大８個の電源オブジェクトのアドレス設定について説明するが、もちろんこの数は８に制限される訳ではない。図８に示すようにすべてのアドレスパルスのあとには固定長１の後ｂ０からｂ７まで８ビット分のアドレス投票区間が用意される。バスラインに接続されているすべての電源オブジェクトは、８ビット長のうちだだ１ビットのみが“１”であるパルス（即ち、データとしては１，２，４，８，１６，３２，６４，１２８のうちの一つ）をランダムに発生し、この区間で同時に投票（出力）する。

【００６３】図９はその様な投票例であり、オブジェクト１から８までがそれぞれ、ｂ６、ｂ４、ｂ５，ｂ３，ｂ２、ｂ０、ｂ０、ｂ７に投票した場合である。従って、これらのオア出力は最下段に示す様に、１１１１１０１となる。さて、各オブジェクトはこのオア出力と自分が投票した出力をたよりにアドレスを設定していく。図９のオア出力をみれば、ＬＳＢ側に一番近いビットで“１”になった場所（この場合はｂ０でＬＳＢそのもの）がわかる。ここに投票したオブジェクトをアドレスｍに設定するため、次のアドレスパルス発生タイミング（この時ｍをインクリメントしてもかまわないが、ここでは簡単のため最終的にアドレスが決定されるまでは、アドレスのインクリメントは行わないとする。つまりアドレスｍが設定完了するまで、アドレスｍがバス管理手段により出力される。）の投票権をこれらのオブジェクトだけに与える（投票権を自分が有しているかどうかは各オブジェクト自身が知る事ができ、しかも他のオブジェクトがどう投票したかについて一切知らなくていい点が重要である）。この場合オブジェクト６と７が投票権を得て、図１０に示す様に、再度投票する。この時

はｂ２、ｂ４にそれぞれ投票し、オア出力は０００１０１００が得られた。ここでオア出力にはまだ２ビット以上“１”が存在するので、以前と同様にＬＳＢ側に一番近いビットで“１”になった場所に対応するオブジェクトが次回の投票権を得る。この投票の様子を示したのが図１１で、ここではオブジェクト６のみが投票をし、オア出力とオブジェクトの投票ビット位置が一致する。この段階で暫定的にオブジェクト６にアドレスｍを付与する。

【００６４】ここで暫定的というのは、各オブジェクトが発生する投票場所は全く確率的なもので、たとえば図９でオブジェクト６、７が同じビットに投票すると、６、７に同じアドレスｍを付与してしまう。そこで、この問題をさけるためのアドレス確認プロセスを入れるが、これについては後述する。

【００６５】さて、確認プロセスも通過して、上記の様に、オブジェクト６のアドレスがｍに確定すると投票権が消滅し、オブジェクト６を除く１から８について、アドレスｍ＋１、ｍ＋２…において同様の操作を行う。これを繰り返す事で、最終的にすべてのオブジェクトのアドレスが確定する。

【００６６】アドレス設定プロセスである事は、バス管理手段がアドレス０のタイミングで発行するデータ中の値で判別する（コマンド＝０ｘ００）。詳細は後述する。

【表１０】

バイト数	宛先アドレス	コマンド
3	ブロードキャスト（０ｘ７ｆ）	０ｘ００

【００６７】（１－２）　アドレス確認プロセス

オブジェクトにアドレスｍが暫定割り当てされたならば、アドレス確認プロセスに入る。このプロセスでは、バス管理手段により発生させられたアドレスｍに対して、それぞれ自分のデータを出力し、データバス上の情報（オアデータ）と自分自身のデータを比較する。（図１２参照）

アドレス確認プロセスであることはバス管理手段のアドレス（０）の後にバス管理手段より発行される以下のデータにより判別できる（コマンド０ｘ０１がアドレス確認プロセスを示す）。（データのフォーマットについては第３部参照。）

【表１１】

バイト数	宛先アドレス	コマンド
3	ブロードキャスト（０ｘ７ｆ）	０ｘ０１

各パケットは、この情報により現在のバスのモードを知る事ができ、暫定アドレスを取得したオブジェクトは、自分の暫定アドレスに続いて自分のデータを出力する。

【００６８】この時のフォーマットはつぎのとおりである。

【表１２】

バイト数	宛先アドレス	コマンド	パラメータ
4	自分のアドレス	０ｘ０１	固有値

宛先アドレスは自分自身のアドレスであり（この様に決めておく）、コマンドはアドレス確認プロセスである事を示す値（０ｘ０１）であるが、パラメータに関しては、自分に固有のものを出力する。例えば、パラメータとしてシリアル番号を出力すれば、二つ以上のオブジェクトの固有データのオアを見ることで、自分以外に同時に出力した別オブジェクトの存在が認識できる。

【００６９】ここで、アドレス重複が検出されたならば、再度アドレス設定プロセスを通過させる。

【００７０】上記（１－１）、（１－２）のアドレス設定、確認プロセスのフローチャートを図１３に示す。なお、図１３の動作はその記述内容から容易に理解できるので詳細な説明は省略する。

【００７１】（２）　データ通信モード

上記（１－１）、（１－２）項のプロセスを経る事で、各電源オブジェクトのアドレスは一義的にきまる。この後バス管理手段プロセッサは、アドレスを定期的に発行し、それに対応するオブジェクトは自分の存在を示すため、データをバスライン上に出力する。これにより、アドレスが一巡すると、バスライン上にあるすべてのオブジェクトは他のオブジェクトの属性とアドレスを共有する。また、このモードではオブジェクト間同士の通信が行われる。この状態をデータ通信モードと定義する。データのフォーマット等詳細については３部で詳述する。

２　オブジェクトの追加、削除（システムの動的再構築）

オブジェクトが電源バスに追加されたときに、そのデータラインを見る事によって、現在他のアドレスがすべて確定しているか、あるいは確定中かがわかる。すべて確定しているならば、空いているアドレス（つまり、アドレスのあとのデータ部分が空）に続けて、自分のデータを出し始めるだけでいい。バス管理手段プロセッサや、他のオブジェクトは、アドレスのあとにデータが入ってきたのを見て、オブジェクトの追加を認識する。確定中ならば、確定するまで待って、その後同様の動作を開始する。

【００７２】これ以降は各オブジェクトが必要に応じて、情報の要求、自発的送信を行う。

【００７３】オブジェクトの削除により、そのアドレスに対応するデータが無くなる（ただし、唯一の電源供給オブジェクトが削除されたときは、システムはそのまま停止する）と、各オブジェクトはこれにより、そのアドレスに付随するオブジェクトの消滅を検出する。この時、管理機能プロセッサはアドレス詰めを行ってもいいし、とりあえずそのアドレスを空きにしておいてもいい（アドレスを現在のオブジェクト数＋１だけポーリングする実装では、アドレス詰めが必須。しかしすべてのアドレスをポーリングする実装ではどちらでもいい）。いずれにしても、オブジェクトの情報はサイクリックにバス上に出力されるので、すべてのオブジェクトは情報の共有が可能である。

【００７４】［第３部　電源バスシステムの制御体系］これまでに、第１部で電源を構成する主要ブロックのオブジェクト化について述べ、第２部で実際の電源システムが如何に構築、維持され、また動的に再構築されるかの概要を説明した。第３部では、これらの構築、維持に使用されるコマンド体系（ＡＰＩ群）およびそのフォーマットについて説明する。

【００７５】なお、以下に示すコード類で総バイト数や宛先アドレスについて、チェックサムを別にしているので注意されたい。

【００７６】１　アドレスフォーマット

各オブジェクトのアドレスはバス管理機能プロセッサよりサイクリックに発生され、その長さは７ビット（従って、最大アドレス数は１２８であるが、バス管理プロセッサのアドレスを０、ブロードキャストアドレスを１２７と定義する）。このアドレスの間に各オブジェクトが自分のデータをバスラインに乗せる事ができる。この部分のフォーマットを図１４に示す。

【００７７】図において１００はバス管理機能プロセッサが発生するアドレスであり、先頭（ＭＳＢ）は常に０、そのあと７ビットのアドレスが続く。なお、このアドレス部はアドレス設定モード以外は、単純にインクリメントされていくのでチェックサムは用意していない。

【００７８】このアドレスの値については次の規定を設ける。

アドレス０（０ｘ００）；これはバス管理機能プロセッサのアドレスと定義する。

アドレス１２７（０ｘ７Ｆ）；これはブロードキャストアドレス。すべてのオブジェクトはこれに続くデータに反応するようになっている。

【００７９】従って、実際にオブジェクトに使用可能なアドレスは１から１２６までの１２６個である。

【００８０】また、バス管理手段が発生するアドレスは０から１２６まですべてサイクリックに発生する方式と、アドレス設定モード内のみ、すべてのアドレスを発生し、データ通信モードでは実際に接続されているオブジェクトの数＋１（つまり、新規追加オブジェクトの追加空きスペース）として周期を短くする方式がある。

【００８１】２　パケットの最大値

１パケットを次の様に定義する。

【表１３】

宛先アドレス（１バイト）	コマンド（１バイト）	パラメータ（１バイト）

このパケットが一つのアドレスのデータ部分に、最小１個（３バイト）、最大１２７個（３８１バイト）存在する。この３バイトを一単位とする総パケット数を各オブジェクトおよびデータバス管理手段プロセッサが発信するデータの先頭に置く。（図１４の１０１）この値を見ることで、何個のコマンドパケットが存在するか分かるとともに、この数でパケットの同期を取る。

【００８２】３　アドレス発生周期の最大値

図１５において１０１以降が、各オブジェクトが発信するデータである。

【００８３】その基本単位は最小３バイトによるパケットであり、パケット数、通信先のアドレス、通信先に対するコマンド、そしてコマンドの引数（パラメータ）より構成される。

【００８４】アドレスで指定されたオブジェクトがいくつのパケット数を送信するかを示すのが、１０１のパケット数表示バイトであり、ＬＳＢをチェックサムとしているため、最小１、最大１２７である。

【００８５】図１６に一サイクルの通信形態と、１００ｋｂｐｓの速度で通信した場合の概略時間を示す。情報パケットの最小は３バイト（パケット数、宛先アドレス、コマンド）であるから、同期のための空白ビット等のオーバーヘッドを無視すると

最小　４ｘ１２７（最大アドレス数）＝５０９バイト

最大　（１＋３ｘ１２７）ｘ１２７＝４８．５Ｋバイト

となり、１００Ｋｂｐｓの通信速度なら、一サイクルは

最小　４１ｍＳ

最大　約３．８秒

となる。最大値は、すべてのアドレスをポーリングし、すべてのアドレスがおのおの１２７バイトのデータを有するときの値であるので、実用的にはこれより十分短い。

【００８６】４　コマンド発生形態

図１７に各オブジェクトからのコマンドの発信状況および、コマンドに対する回答状況を示す。すべてのコマンドの発信とその回答は１対１になる様に設計され、必ずハンドシェークとなる。アドレスの発生周期１、２、３はこの順に連続している。例えばオブジェクト１（アドレス１をアサインされたオブジェクト）がｎに対してコマンドを発行（２０４）すると、オブジェクトｎはこの内容を発行直後に（つまり、例えば２０６のタイミングまで待たずに）認識できる。そこでこれに対する回答を用意しておき、自分のデータ発信タイミング（これが２０６）で実際に情報発信を行う。また、オブジェクトｎ

25

は複数のアドレスにデータを送信できるので、例えばオブジェクト２に対する回答（２０７）やオブジェクト３に対するコマンド発信（２０８）が、自分の割り当てタイミング（つまりアドレスｎを受け取ったあと）の中で可能である。

【００８７】命令や情報要求（これらはすべてコマンドと総称する）に対する、動作完了や回答時に使用するコマンドは、返答コマンドとして定義した。（コード０ｘ２６）

また、同一宛先に対して複数の命令を送る事も可能で、この時は宛先アドレス－コマンド－パラメータのシーケンスを繰り返し、また回答はコマンド順に発生すると定義する。（２１２、２１３）

コマンドに対する回答のペアはネストを禁止してあり、コマンド－回答の間でエラーが発生した場合には、再度コマンドを発行する等で対処する。また、コマンドを自

26

分自身に対して発行する事も可能であり、例えば負荷オブジェクトをオンするコマンドを負荷オブジェクト自身が発行すれば、バスラインに接続されている他のオブジェクトもこの情報が共有できる。これにより電源供給源オブジェクトは、電流が増えた事を検出せずに、負荷が接続されて事が知れる。

【００８８】５　バスのモード電源バスには上述した様に、アドレス設定モードとデータ通信モードがあるが、これの違いはバス上で以下の様に表明される。

【００８９】アドレス０はバス管理機能プロセッサのアドレスであり、その後に最小３バイトのデータパケットが存在する。（この形式は他のオブジェクトと同じ）。

【００９０】このデータパケットを用いて、上記モードの表明を行う。

【表１４】

	バイト数	宛先アドレス	コマンド
アドレス設定モード			
アドレス設定プロセス	3	ブロードキャスト	０×００
	(0x03	0x7f	0x00)
アドレス確認プロセス	3	ブロードキャスト	０×０１
	(0x03	0x7f	0x01)
データ通信モード	3	ブロードキャスト	０ｘ７ｆ
	(0x03	0x7f	0x7f)

これらのモードの遷移は一サイクル（すべてのアドレスポーリング完了まで）単位となる。

【００９１】６　オブジェクト属性発信体系

電源バスがデータ通信モードになると、各オブジェクトは自身の属性をブロードキャストする。この時のデータの形式は以下の様になる。

【表１５】

宛先アドレス	コマンド	パラメータ	
0x7f	0x20	0x01	Generic Power Source
		0x02	Generic Charger
		0x03	AC adaptor
		0x04	DC adaptor
		0x05	NiCd battery
		0x06	NiH battery
		0x07	Li ion battery
		0x08	Lead battery
		0x09	Other battery
		0x10	Generic Temp Power Storage
		0x20	Generic Generator
		0x80	Generic Load

各オブジェクトはアドレスとそれに付随するコマンド、パラメータを読む事により、現在の電源バスライン上にどのような属性のものが接続されているかが知れる。さらに詳しい情報は必要なオブジェクトが相手先に問い合わせ、認識できる。

【００９２】７　コマンド体系および使用例

本明細書はコマンド体系全体についての仕様書ではないので、いくつかの使用例におけるコマンド（ＡＰＩ）群について記述する。なお、ここでいうコマンドは広い意味であり、内容的には情報要求命令、要求情報回答、動作命令、モード表明等を含んでいる。

【００９３】（７－１）データバス管理コマンド

データバス管理コマンドとしてはつぎのものがある。

【表１６】

return value	API	Code	Parameter	Description
void	AdSetProces	0x00	padding	Address is under negotiation
void	AdConfirmProces	0x01	padding	Address is under confirmation
void	DataComMode	0x02	padding	Data communication mode

【００９４】（７－２）データ通信モード中に使用されるコマンド

データ通信モード中に使用されるコマンドとしてはつぎのようなものがある。

【表１７】

return value	API	Code	Parameter	Description
BlockType	BlockTypeInq	0x10	padding	Inquiry for power block type
PowerMode	ModeInq	0x11	padding	Inquiry of power source or load
boolean	SwitchCont	0x13	boolean	Switch on/off
boolean	SwitchInq	0x14	padding	Inquiry of switch on/off
PowerType	PowerType	0x15	padding	Inquiry of power source type
int	CatalogVoltage	0x16	padding	Inquiry of nominal voltage
int	MaxVA	0x17	padding	Inquiry of maximum VA
int	AlarmVoltage	0x18	padding	Inquiry of alarm voltage
int	OutputVoltage	0x19	padding	Inquiry of output voltage
int	OutputCurrent	0x1a	padding	Inquiry of output current
int	CurrentImp	0x1b	padding	Inquirey of output impedance
int	ServiceTime	0x1c	padding	Inquiry of expected service time
boolean	IsUsable	0x1d	pointer to Load	Inquiry of power source usable
boolean	SetMinServTime	0x1e	time	Set minimum service time
int	RemainVA	0x1f	padding	Inquiry of remained power
int	GetLifeTime	0x20	padding	Inquiry of remained lifetime
int	Capacitance	0x21	padding	Inquiry of capacitance
int	Multiplier	0x22	padding	Inuiry of unit multiplier
int	SetVoltLim	0x23	voltage	Set limit voltage
int	SetCurrLim	0x24	current	Set limit current
boolean	ChargeCont	0x25	boolean	Charge on/off control
int	DataReply	0x26	reply data	Reply data

【００９５】（７－３）例外コマンド

本アーキテクチャのフォーマットに準拠しない、固有のデータやコマンド等を通信するためのラッパとして次のものを定義する。

【表１８】

return value	API	Code	Parameter	Description
**	Wrapper	0x70	**	Command wrapper
**	ReplyWrapper	0x71	**	Reply wrapper

以下、実際の応用例である

【表１９】

パケット数	宛先アドレス	コマンド	パラメータ
m	1バイト	０x７０	固有情報
n	1バイト	０x７１	固有リプライ情報

パラメータ中に固有情報を挿入するが、その長さはmパケット（３mバイト）となるように調節し、最大１２７パケット（３ｘ１２７バイト）である。

【００９６】（７－４）データ通信モード中での他のオブジェクトに対する情報要求例

（第１部４－１参照）負荷オブジェクト（アドレス８）がバスライン上の電源供給オブジェクト（アドレス２）に対し、その詳細情報要求例を示す。

【表２０】

address	bytesize	destination address	command	parameter
0x08	0x04	0x02	0x14	0x00(padding)
0x02	0x04	0x08	0x15	0x00

ここに、コマンド０ｘ１４はオブジェクトに対するスイッチデータ情報要求、パラメータ０ｘ００は単にスペースを埋めるためのもの、電源供給オブジェクトが発行したコマンドは０ｘ１５でコマンドに対する返事を表現し、パラメータ０ｘ００は、スイッチがオフである。

【００９７】８　クラス一覧以下、クラスの一覧を示す。

【００９８】（８－１）基本クラス

基本クラスはつぎのように記述される。

【表２１】

29 30

```
class PowerBlock {
    private :
       BlockType block_type ;
       PowerMode power_mode ;
       int address ;
       boolean switch ;
    public :
       BlockType BlockTypeInq(){  return block_type ; }
       PowerMode ModeInq(){ return power_mode ; }
       boolean SwitchCont( boolean cont){ return switch ; }
       boolean SwitchInq(){  return switch ; }
};
```

【００９９】（８－２）電源供給オブジェクト　【表２２】

電源供給オブジェクトはつぎのように記述される。

```
class PowerSource : public PowerBlock {
   private:
      PowerType power_type ;
      int catalog_voltage ;
      int max_current ;
      int max_va ;
      int alarm_voltage ;
      int output_voltage ;
      int output_current ;
      int current_imp ;
      int service_time ;
      int min_service_time ;
      LoadList *load_list ;
   public:
      PowerType PowerType(){ return power_type ; }
      int CatalogVoltage(){ return catalog_voltage ; }
      int MaxVA(){ return max_va ; }
      int AlarmVoltage(){ return alarm_voltage ; }
      int OutputVoltage(){ return output_voltage ; }
      int OutputCurrent(){ return output_current ;}
      int CurrentImp(){ return current_imp ; }
      int ServiceTime(){ return service_time ; }
   public :
      boolean IsUsable( Load *l) { ; }
      boolean SetMinServTime( int t) {   }
      int RemainVA() {   }
```

【０１００】（８－３）チャージャ　【表２３】

チャージャはつぎのように記述される。

31 32

```
class Charger :public PowerSource{
    private :
       int charger_type ;
       int max_output_voltage ;
       int max_output_curr ;
       int voltlim ;
       int currlim ;
       boolean charge ;
    public :
       int SetVoltLim( int v ){ return voltlim ; }
       int SetCurrLim( int c ){ return currlim ; }
       boolean ChargeCont( boolean cont ){ return charge ;}
} ;
```

【０１０１】（８－４）負荷　　【表２４】

負荷はつぎのように記述される。

```
Class Load : public PowerBlock {
   private:
      LoadType load_type ;
      int max_voltage ;
      int min_voltage ;
      int max_curent ;
      int min_current ;
      int suspend_curr ;
      LoadMode load_mode ;
      int input_voltage ;
      int input_current ;
   public:
      LoadType LoadTypeInq(){ return load_type ;}
      int MaxVoltage(){ return max_voltage ;}
      int MinVoltage(){ return min_vltagae ;}
      int MaxCurrent(){ return maz_current ;}
      int MinCurrent(){ return min_current ;}
      int SuspendCurrent(){ return suspend_current ;}
      LoadMode LoadModeInq(){ return load_mode ;}
      int InputVoltage(){ return input_voltage ;}
      int InputCurrent(){ return input_current ;}
      int GetVA() {    ;}
      int LoadCont( int cont){  ;}
      int GetCurrPowerConsmpt(){  ;}
} ;
```

【０１０２】（８－５）二次電池　　【表２５】

二次電池はつぎのように記述される。

33　　34

```
class RechargeBatt : public PowerSource, public Load {
    private:
       int ModelName ;
       int TotalLife ;
       int TotalUsedTime ;
    public :
       int GetLifeTime() ;
} ;
```

【０１０３】（８－６）一時電源
一時電源はつぎのように記述される。

【表２６】

```
class TempStorage : public PowerSource, public Load {
   private :
     int capacitance ;
     int multiplier ;
   public :
     int Capacitance(){ return capacitance ;}
     int Multiplier(){ return multiplier ;}
} ;
```

つぎに、電源および負荷が１つずつの最も基本的な例を用いて具体的に説明する。
【０１０４】図１８において、バスコントローラ３０１に電源バスライン３０２、ＧＮＤバスライン３０３およびデータバスライン３０４が接続されている。これらバスラインにコネクタ３０７および３０８を介して電源３０５および負荷３０６がそれぞれ接続されている。
【０１０５】各部の動作状況を説明する。
【０１０６】（１）電源の内容
ＰｏｗｅｒＳｏｕｒｃｅクラスに属する電源の仕様は以下のものであるとする。
ｐｏｗｅｒ＿ｔｙｐｅ＝　０ｘ００　（二次電池でさらにＮｉＣｄ電池であることを示す）
ｃａｔａｌｏｇ＿ｖｏｌｔａｇｅ＝６　（公称出力電圧＝６Ｖ）
ｍａｘ＿ｖａ＝６　（公称容量＝６ＶＡ）
ｏｕｔｐｕｔ＿ｖｏｌｔａｇｅ＝７　（現在の出力電圧＝７Ｖ）
ａｌａｒｍ＿ｖｏｌｔａｇｅ＝５．６　（最低供給可能電圧＝５．５Ｖ）
上記データの中で、現在出力電圧が公称出力電圧より高いが充電直後はこの様になるのが普通である。また最低供給可能電圧５．５Ｖは、例えば、これより低い電圧になるまで電池が使用されると電池が痛むという値である。
【０１０７】（２）負荷の内容
一方ｌｏａｄクラスに属する負荷のデータは以下のものと仮定する。
ｍａｘ＿ｖｏｌｔａｇｅ＝８Ｖ　（入力許容最大電圧＝８Ｖ）
ｍｉｎ＿ｖｏｌｔａｇｅ＝４Ｖ　（動作可能最小電圧＝４Ｖ）
ｍａｘ＿ｃｕｒｒｅｎｔ＝１Ａ　（入力最大電流＝１Ａ）
【０１０８】（３）電源システムの動作
図１８に示すようなブロックで、電源３０５がバスに接続された後、負荷３０６がバスに接続される場合を想定する。（負荷が先に接続されても、何も起こらない）
図２に示すように電源オブジェクトの中のマイクロプロセッサ１９はＤＣ電源１０がある値以上であればつねに動作している。
【０１０９】従って電源３０５がバスに接続されると、バス管理手段（バスコントローラ３０１）が動作を開始し、まずアドレス設定モードでシステムが動作する。これにより図１３に示すフローチャートに基づき、電源オブジェクト（電源３０５）のアドレスが決定する（アドレス１となる）。
【０１１０】アドレスが決定された時点で、このバスには他に電源が接続されていない事（実際には他には何も接続されていない事）がわかるので、バスシステムを使用可能とするため、電源オブジェクトのメインスイッチ（１２）を投入する。
【０１１１】つぎに負荷オブジェクト（負荷３０６）が接続されると、再度アドレスの調停を行い、負荷オブジェクトのアドレスが決定（アドレス２）する。
【０１１２】以上のプロセスを経て、電源バスシステム

は動作準備が整う。
（４）実際のオペレーション例
電源バスシステムがどの様に使われるかは本発明の趣旨であるアーキテクチャの上のアプリケーションプログラムで決定されるものであるが、以下に一例をしめす。
【０１１３】負荷として上記の様に（上記（２）項）仮定したが、実際の状況ではこの負荷の先にラジオであるとか、ＰＤＡであるとかの製品が存在する。ここでは一番簡単にするため、オンオフ機能のみ付いている製品で例えばフラッシュライトとする。
【０１１４】フラッシュライトの場合の負荷オブジェクトの構造を図１９に示す。３０９がマイクロプロセッサ、３１０はメインスイッチで、例えばＭＯＳＦＥＴ、３１２はランプ、３１１がランプのコントロールスイッチ、３１３はアラーム表示ＬＥＤである。
【０１１５】コネクタ３０８がバスシステムに接続され、負荷オブジェクトのアドレス調停が完了した時点で、マイクロプロセッサは次の様な動作を始める。この動作を図２０に示す。なお、図２０の動作は基本的に図から容易に理解できるので、ここでは簡単に説明を行うのみとする。
【０１１６】電源電圧の問い合わせ；ＣａｔａｌｏｇＶｏｌｔａｇｅ（）関数を使用して電源オブジェクトに対して公称出力電圧を問い合わせ、これと自分自身の最大許容電圧を比較し、電源電圧が適正かどうか判断する。公称電圧はもちろん実際電圧とは異なるが、これがわかればおよその判断は可能である。もし、非常に厳密な判断が必要なら、そのようなデータや問い合わせＡＰＩを定義すればいい。
【０１１７】電源の種類問い合わせ；電源電圧は適正でも自分が必要な電流容量を判断するために、電源の種類から判断する。フラッシュライトで最大１Ａならば、ほとんどの一次、二次電池は使用可能である。これも電流容量が正確に知りたいならばそのようなＡＰＩを用意すればいい。
【０１１８】ここまでで、電源の電圧とおよその供給電流が知れたので、フラッシュライトのコントロールスイッチを検出する。ここでスイッチがオンされちならば、電球１１２に電力を供給する。この後はバッテリー残量を適宜問い合わせ（フローチャートでは、この辺の時間のウェイトをかける部分は省略）、バッテリー残量が少なくなったならば、電球を切り、アラームＬＥＤを点灯する。
【０１１９】アラームＬＥＤは、フラッシュライト使用中にバッテリが少なくなった時のみでなく、バスシステムに接続されている電源との相性が悪いときにも点灯させて、ユーザに状況を表示するためのものである。
【０１２０】
【発明の効果】以上説明したように、この発明によれば、以下のような効果を実現できる。１モバイル機器、ウエアラブル機器に対して、共通の電源管理アーキテクチャを提供し、バッテリや電源の共通化がはかれる。
２　異なった種類の電源（一次電池、二次電池、ＡＣアダプター、ＤＣアダプター、発電機）や異なった機器（負荷）の間で、お互いの状況を通信する共通の通信手段および言語体系を提供し、複数の機器間で共通に使用可能な電源関連機器設計ができる。
３　各種の電源供給源を共通のアーキテクチャの中で定義する事により、幅広いエネルギー供給源を機器の電源に使用可能となり、機器使用環境が飛躍的に拡大する。
４　複数の異なった種類の電源が共通のバスラインに論理的に並列接続が可能となり、バスラインに接続する電源の数を増やすだけで、電源容量を上昇できる。
５　本電源管理アーキテクチャはスケーラブルであり、比較的小電力のものから、オブジェクト数が多く、かつ大電力を扱うものまで同一の考えで設計できる。
６　本アーキテクチャに基づいて設計する事により、電源関係ブロックの標準化が可能で、電源システム設計の大幅な省力化が可能である。
７　オブジェクトエミュレータにより、例えばＰＣなどにより電源の動作状態をモニターする事ができ、信頼性の高い電源システムを容易に構築できる。
【図面の簡単な説明】
【図１】　この発明の実施例の構成を全体として示す図である。
【図２】　上述実施例の電源オブジェクトの構成例を説明する図である。
【図３】　上述実施例の一時エネルギーストレージオブジェクトの構成例を説明する図である。
【図４】　上述実施例の負荷オブジェクトの構成例を説明する図である。
【図５】　上述実施例のバス管理手段を説明する図である。
【図６】　上述実施例のアドレス付与を説明する図である。
【図７】　上述実施例のアドレス付与を説明する図である。
【図８】　上述実施例のアドレス付与を説明する図である。
【図９】　上述実施例のアドレス付与を説明する図である。
【図１０】　上述実施例のアドレス付与を説明する図である。
【図１１】　上述実施例のアドレス付与を説明する図である。
【図１２】　上述実施例のアドレス付与を説明する図である。
【図１３】　上述実施例のアドレス付与を説明する図である。
【図１４】　上述実施例のデータ通信フォーマットを説

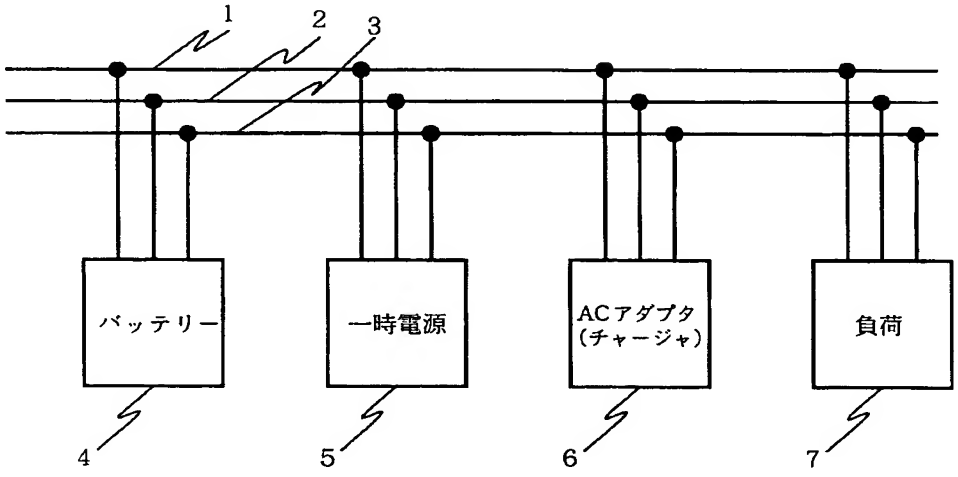
37

明する図である。
【図１５】　上述実施例のデータ通信フォーマットを説明する図である。
【図１６】　上述実施例のデータ通信フォーマットを説明する図である。
【図１７】　上述実施例のコマンドの発信、回答を説明する図である。
【図１８】　上述実施例を具体例をあげて説明するずである。
【図１９】　図１８の負荷の構成を説明する図である。
【図２０】　図１８および図１９の動作を説明する図である。
【符号の説明】
１　電源バスライン
２　ＧＮＤ（接地）バスライン
３　データ（信号）バスライン
４　バッテリー
５　一時電源
６　ＡＣアダプタ
７　負荷（一般電気機器）
１０　ＤＣ電源

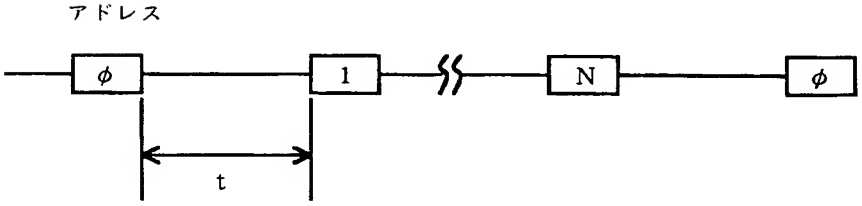
38

１１　電流検出用抵抗
１２　スイッチ
１３、１４、１５、２１　マイクロプロセッサの入出力
１６　電力バスライン
１７　情報バスライン
１８　接地バスライン
１９　マイクロプロセッサ
２０　電気二十層コンデンサ
５０　スイッチ
５１　消費電流検出用抵抗
５２、５３、５４　マイクロプロセッサ５５の入出力
５５　マイクロプロセッサ
５６　通信路
５７　本来の負荷
５８　負荷５７のマイクロプロセッサ
５９　コンデンサ
６０　マイクロプロセッサ
６１　プルアップ抵抗
６２　データ出力用トランジスタ
６３　バス管理手段（バスコントローラ）
６４　電源オブジェクト

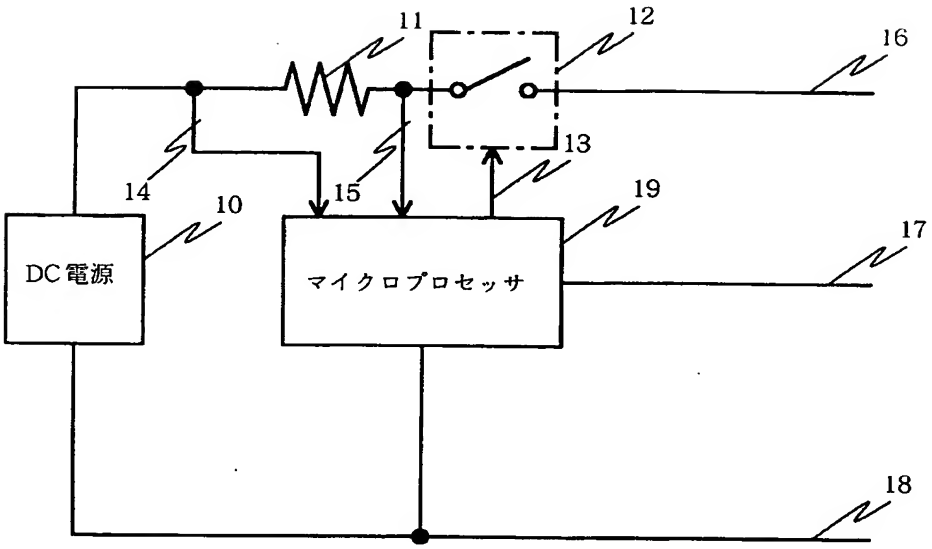
【図１】



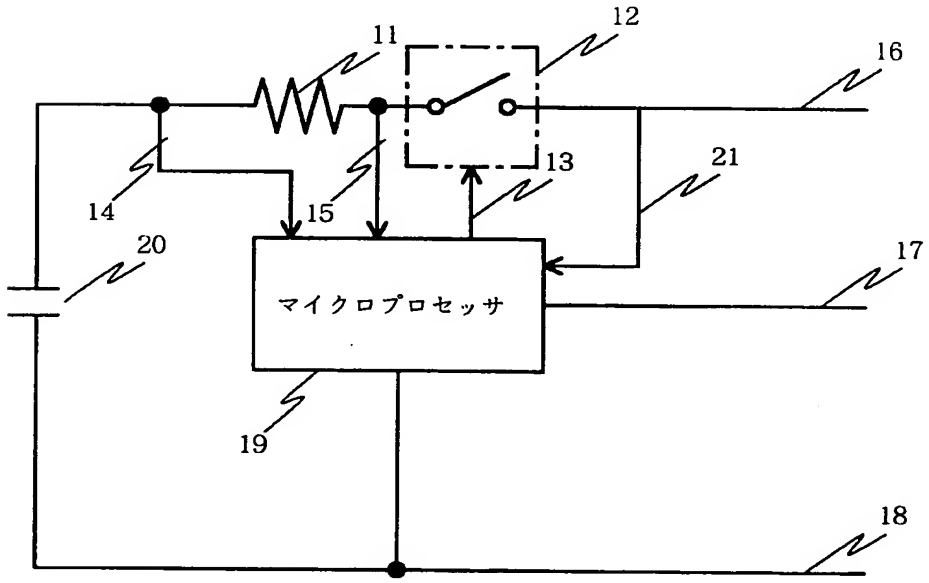
【図６】



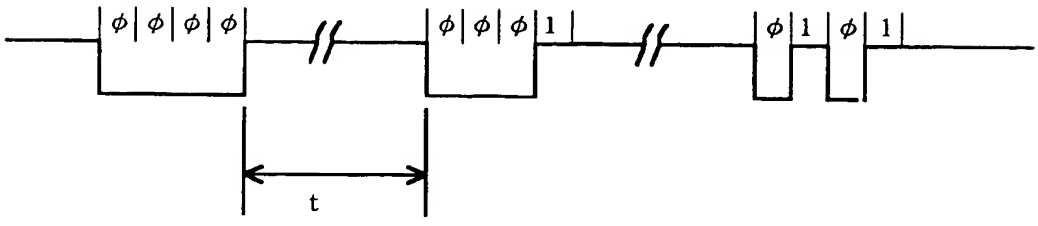
【図２】


11
12
16
13
14
15
19
10
17
DC電源
マイクロプロセッサ
18

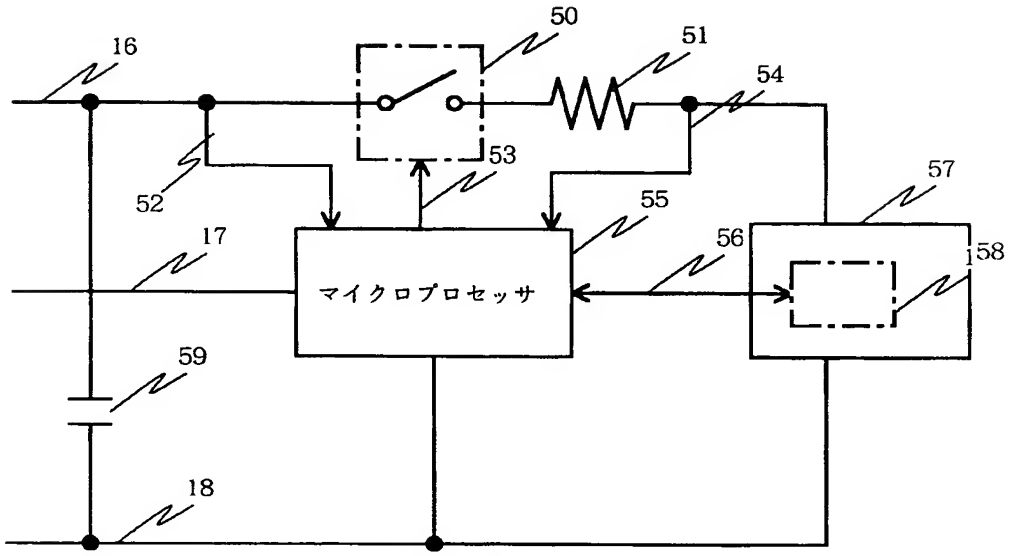
【図３】


11
12
16
13
21
14
15
20
17
マイクロプロセッサ
19
18

【図７】

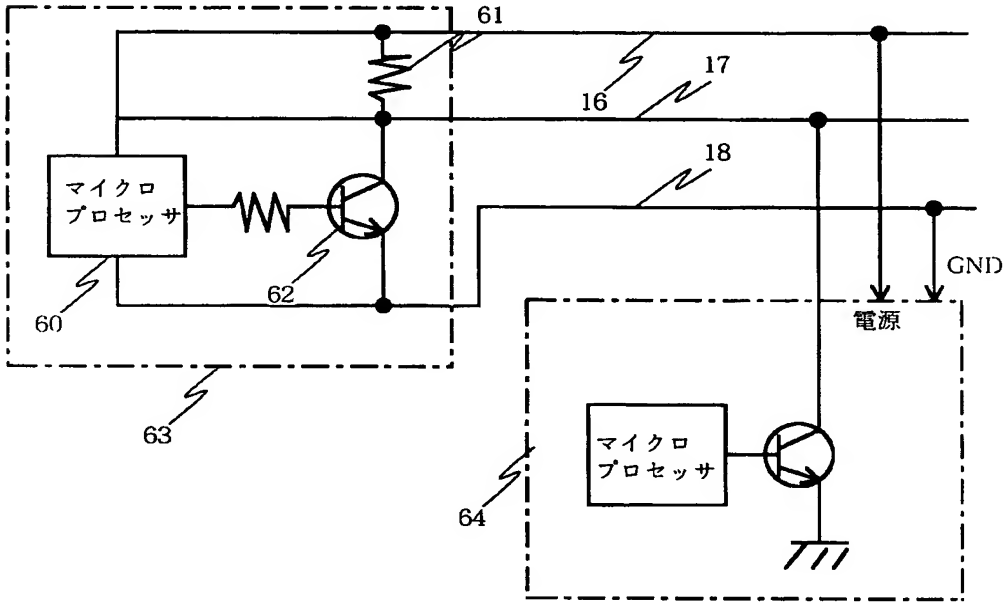

φ φ φ φ
φ φ φ 1
φ 1 φ 1
t

【図４】

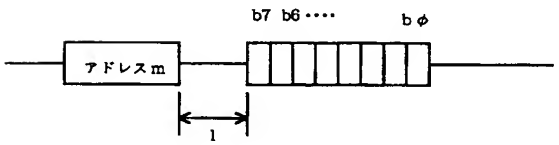

16
50
51
54
52
53
57
55
17
56
58
マイクロプロセッサ
59
18

【図５】

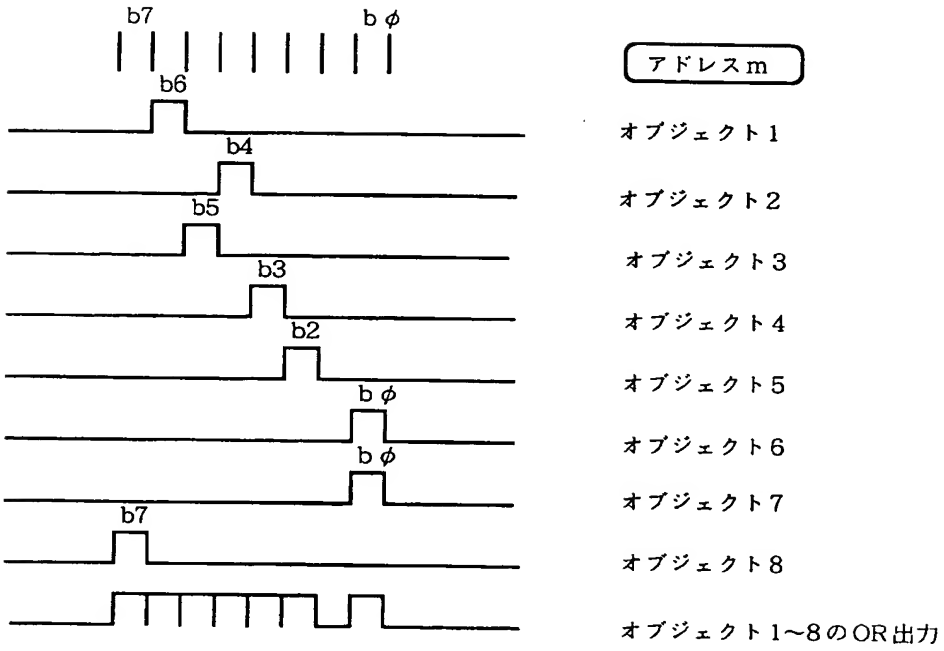
【図５】


61
16
17
18
マイクロ
プロセッサ
GND
62
60
電源
63
マイクロ
プロセッサ
64

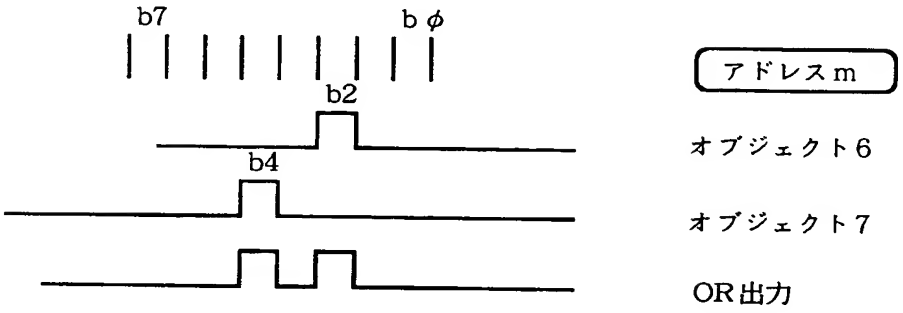
【図８】


b7 b6 ・・・・
b０
アドレスm
1

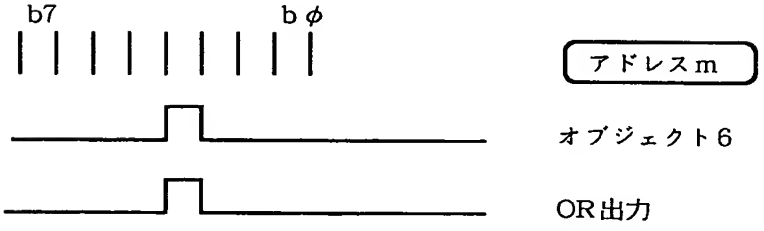
【図９】


b7
bφ
アドレスm
b6
オブジェクト1
b4
オブジェクト2
b5
オブジェクト3
b3
オブジェクト4
b2
オブジェクト5
bφ
オブジェクト6
bφ
オブジェクト7
b7
オブジェクト8
オブジェクト1～8のOR出力

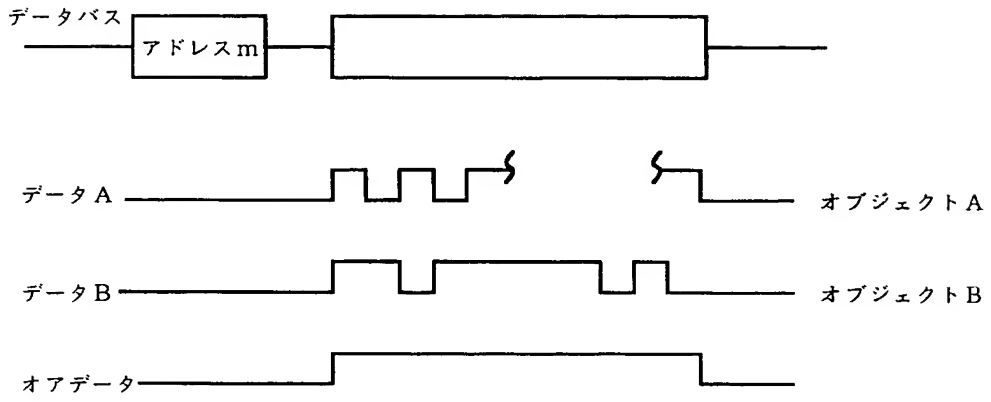
【図１０】


b7
bφ
アドレスm
b2
オブジェクト6
b4
オブジェクト7
OR出力

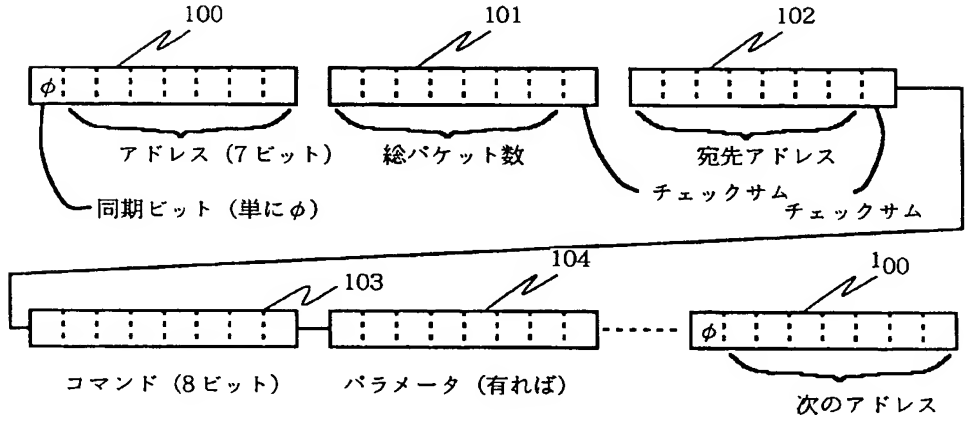
【図１１】


b7
bφ
アドレスm
オブジェクト6
OR出力

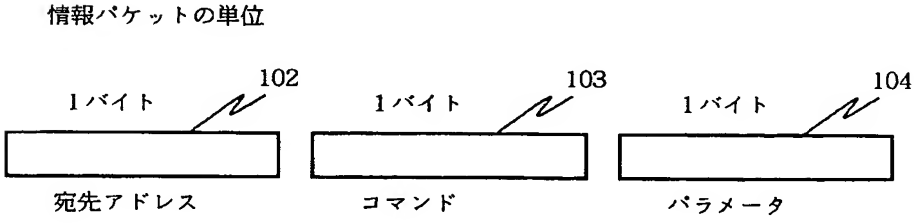
【図１２】


データバス
アドレスm
データA
オブジェクトA
データB
オブジェクトB
オアデータ

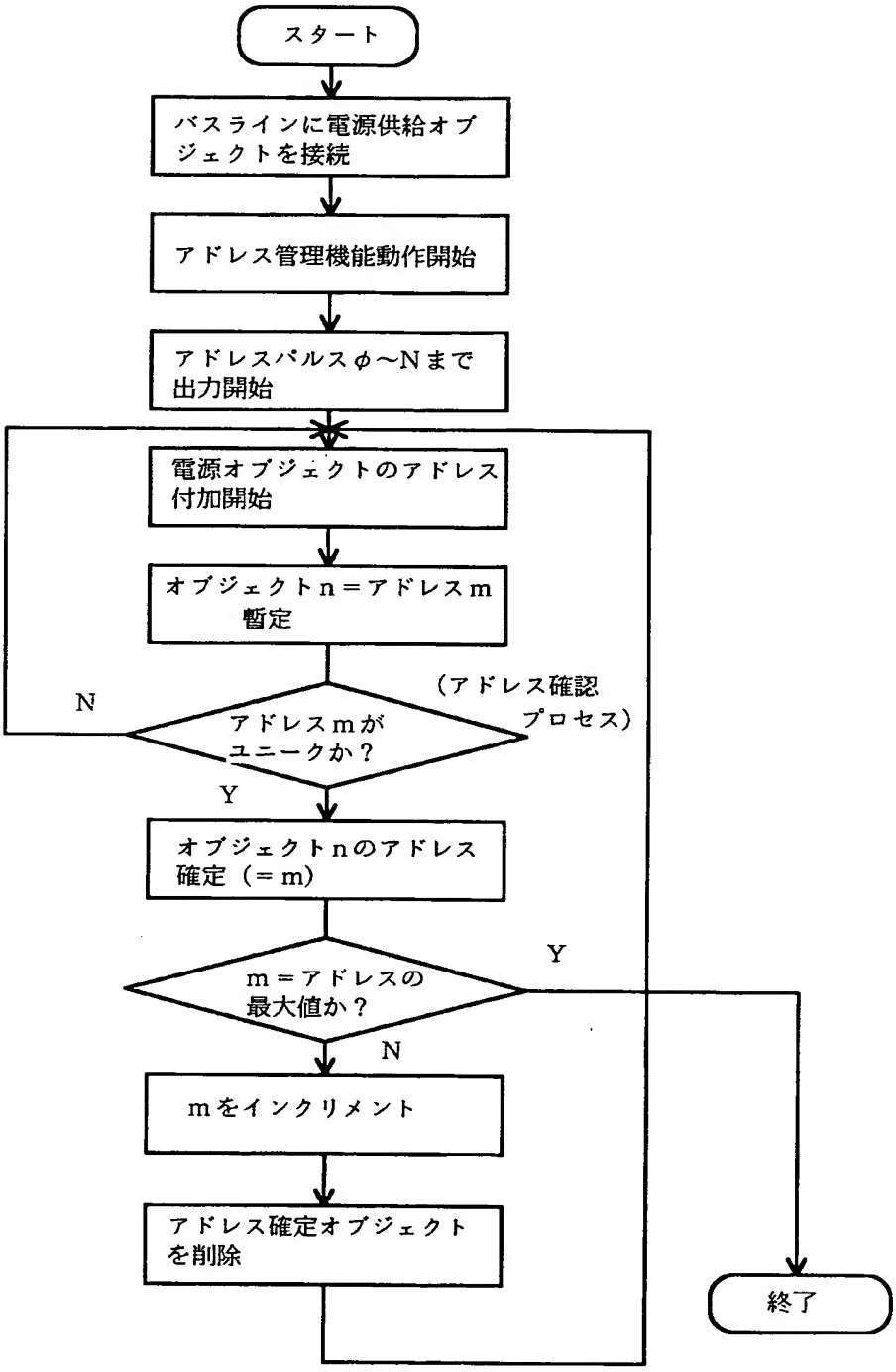
【図１４】


100
101
102
φ
アドレス（7ビット）
総パケット数
宛先アドレス
同期ビット（単にφ）
チェックサム
チェックサム
103
104
100
φ
コマンド（8ビット）
パラメータ（有れば）
次のアドレス

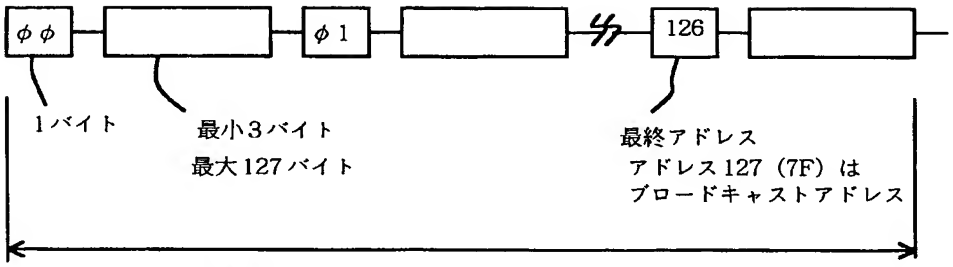
【図１５】


情報パケットの単位
1バイト
102
1バイト
103
1バイト
104
宛先アドレス
コマンド
パラメータ

【図１３】


スタート
バスラインに電源供給オブジェクトを接続
アドレス管理機能動作開始
アドレスパルスφ～Nまで出力開始
電源オブジェクトのアドレス付加開始
オブジェクトn＝アドレスm暫定
（アドレス確認プロセス）
アドレスmがユニークか？
N
Y
オブジェクトnのアドレス確定（＝m）
m＝アドレスの最大値か？
Y
N
mをインクリメント
アドレス確定オブジェクトを削除
終了

【図１６】



最小4 × 127 = 509バイト

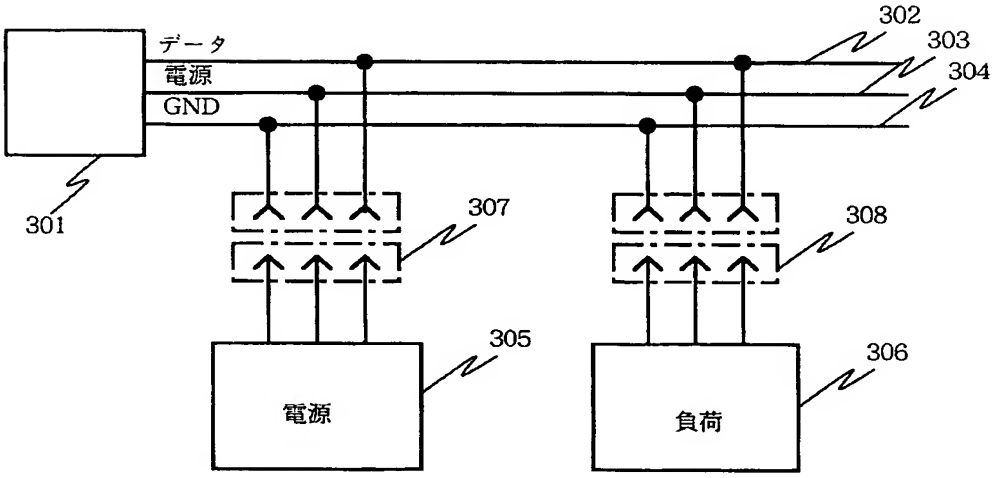
最大127 × 127 = 1601Kバイト

100kbpsの場合

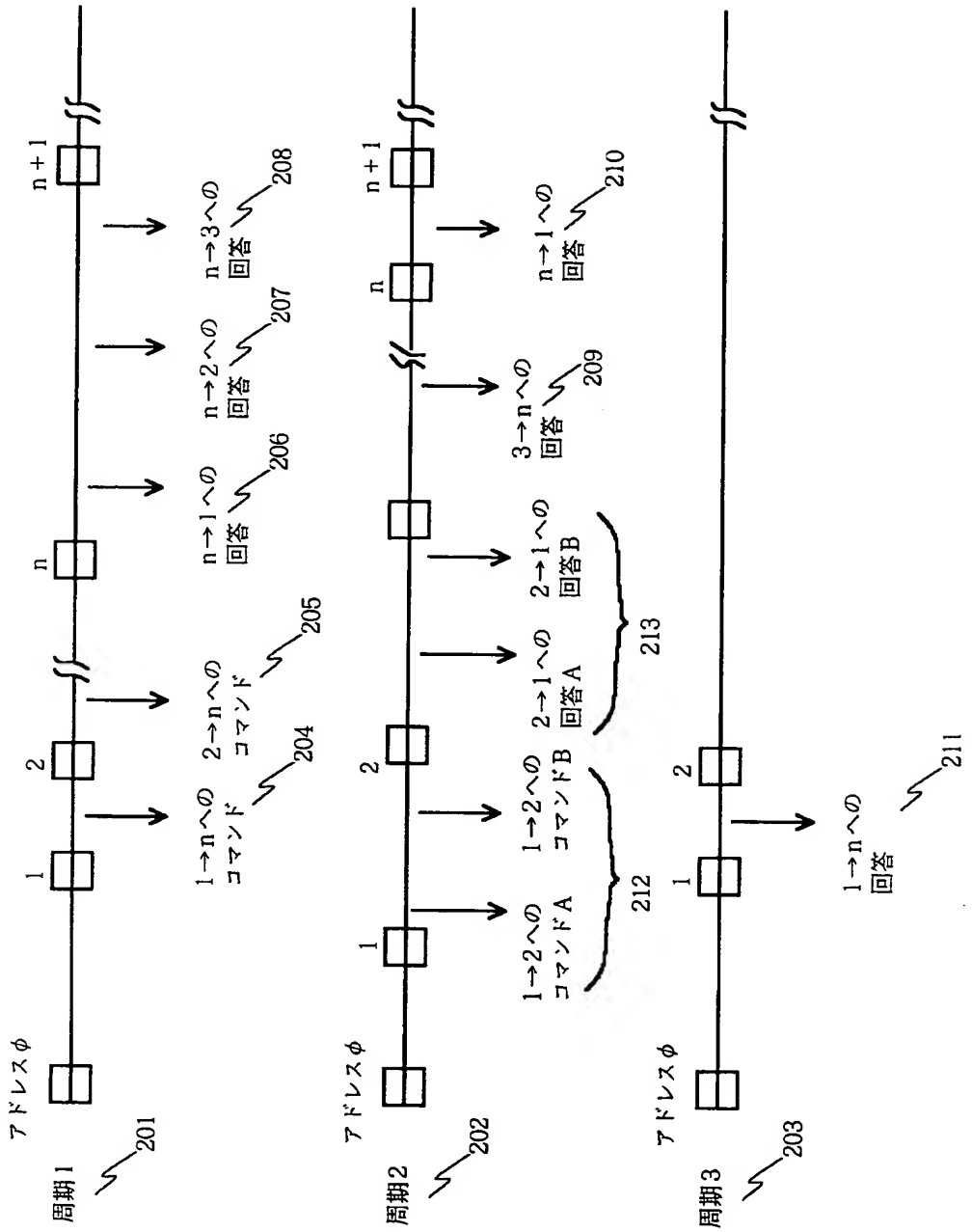
最小41mS

最大1290mS（≒1.3秒）

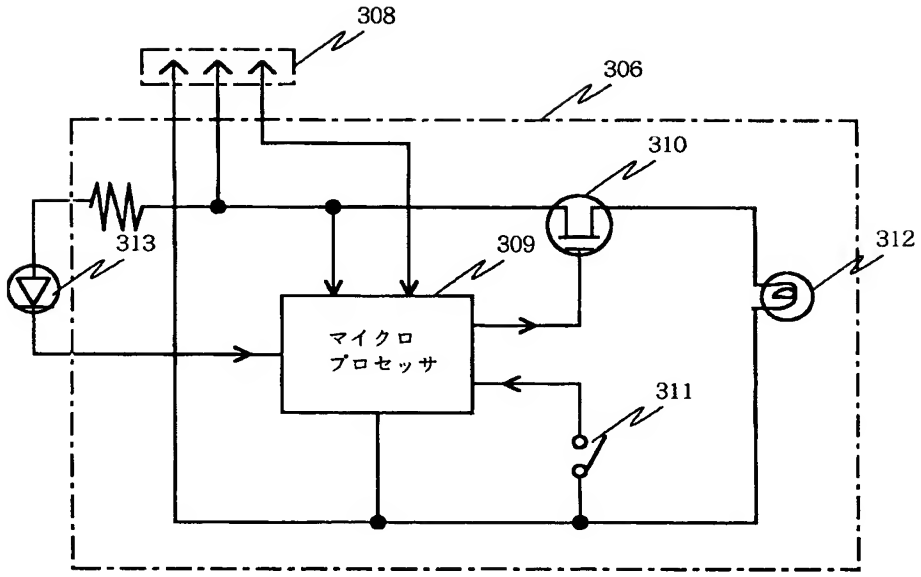
【図１８】



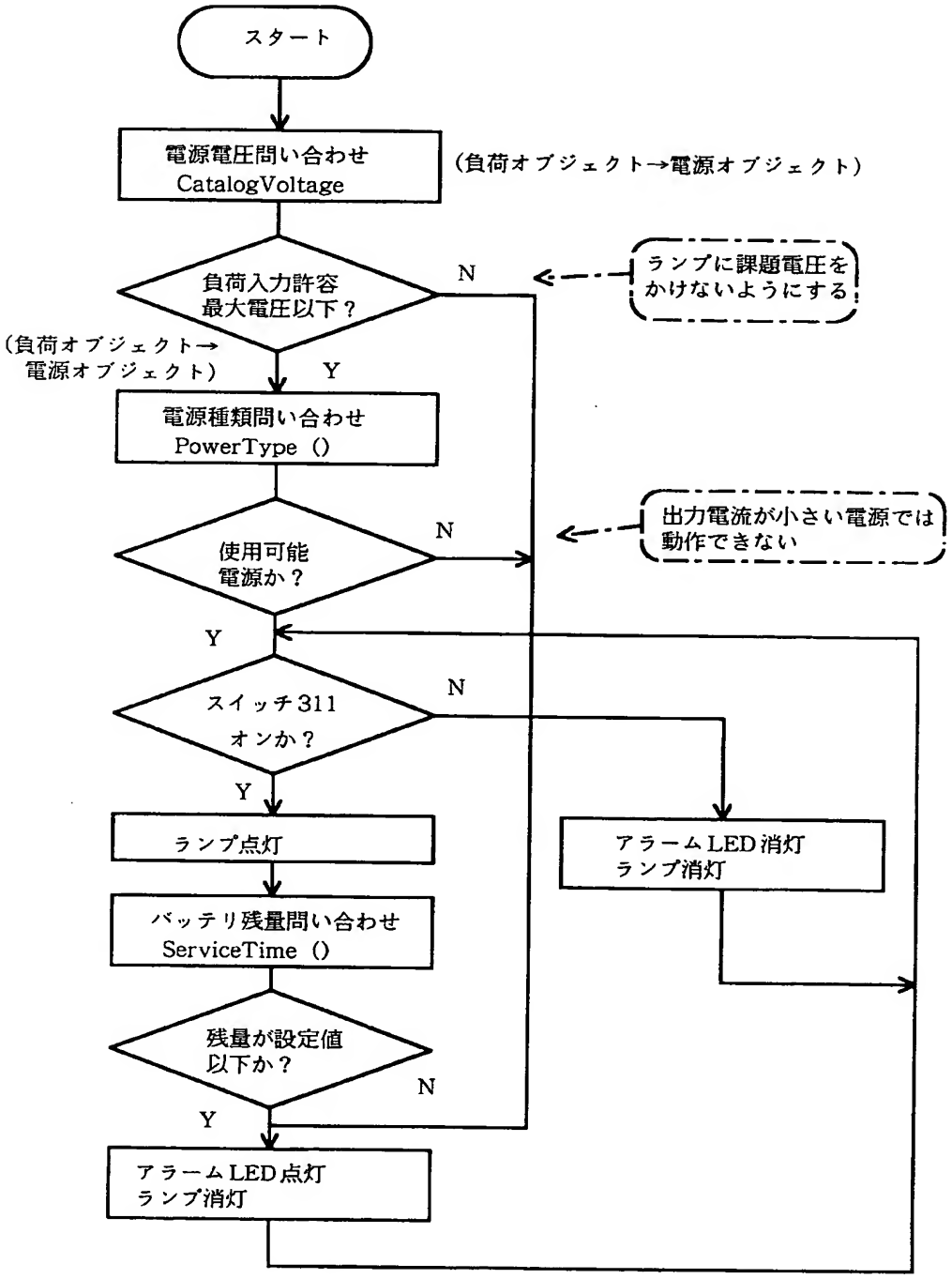
【図１７】


アドレスφ
周期1
201
1
2
n
n+1
1→nへの コマンド
204
2→nへの コマンド
205
n→1への 回答
206
n→2への 回答
207
n→3への 回答
208
アドレスφ
周期2
202
1
2
n
n+1
1→2への コマンドA
212
1→2への コマンドB
2→1への 回答A
2→1への 回答B
213
3→nへの 回答
209
n→1への 回答
210
アドレスφ
周期3
203
1
2
1→nへの 回答
211

【図１９】



【図２０】


スタート
電源電圧問い合わせ
CatalogVoltage
(負荷オブジェクト→電源オブジェクト)
負荷入力許容
最大電圧以下？
N
ランプに課題電圧を
かけないようにする
(負荷オブジェクト→
電源オブジェクト)
Y
電源種類問い合わせ
PowerType ()
使用可能
電源か？
N
出力電流が小さい電源では
動作できない
Y
スイッチ311
オンか？
N
Y
ランプ点灯
アラームLED消灯
ランプ消灯
バッテリ残量問い合わせ
ServiceTime ()
残量が設定値
以下か？
N
Y
アラームLED点灯
ランプ消灯